Accountable Light Client Systems for PoS Blockchains

Oana Ciobotaru , Alistair Stewart , and Sergey Vasilyev

Web3 Foundation

August 17, 2022

1 Introduction

Blockchain systems rely on consensus among a large number of participants. Following consensus of a blockchain network can become expensive in terms of networking bandwidth, storage and computation. Depending on the type of consensus these challenges can be aggravated when the size of participants is large or when the set of participants changes frequently. Light clients (such as SPV clients in Bitcoin or inter-blockchain bridge components that support interoperability) are designed to allow users to follow consensus of a blockchain with minimal cost.

We are interested in blockchains that use Byzantine agreement type consensus protocols, particularly proof of stake systems like Polkadot [1], Ethereum 2 [**?**] or many other systems [2, 3, 4, 5, 6]. These protocols may have large numbers of consensus participants, from 1000s to 100000s and in such proof of stake protocols, the set of participants may change regularly.

Following the consensus of such blockchain protocols systems in the examples above boils down to proving that a large subset of a designated set of participants, called validators, signed the same message (e.g. a block header). There are a number of obvious approaches to verifying that a subset of a large set signed something.

Verify All Signatures One could verify a signature from each signing validator. This is what participants of protocols like Polkadot [1] or Tendermint [5, 7] do, however it becomes prohibitively expensive for a light client when there are of thousands (even more so where there are tens of thousands) of signatures.

Aggregatable Signatures One could use an aggregatable signature scheme like BLS [8, 9] and reduce this to verifying one signature, but that requires calculating an aggregate public key. This aggregate key is different for every subset of signers and needs to be calculated from the public keys. This is what Ethereum 2 [**?**] does. However for a light client, it is expensive to keep a list of thousands of public keys updated.

Threshold Signatures It would be possible to have a threshold signature scheme, with one public key for the entire set. This approach was adopted by Dfinity [10]. Threshold signatures have two downsides. Firstly, it requires a communication-heavy distributed key generation protocol for the setup which is difficult to scale to large numbers of validators and may require being performed every time the validator set changes. Secondly, we cannot tell which subset of the validators signed a signature, which we will discuss below.

Our Approach: Committee Key Scheme Our approach is to extend an aggregatable signature scheme with a commitment to a list of public keys[1] and with proofs for aggregating subsets of public keys from the

[1] In our particular use case, each of the public keys belongs to a validator that works towards achieving consensus e.g., on a blockchain.

list. We call this extension a committee key scheme for aggregatable signatures or, for short, a committee key scheme. In more detail, the committee key scheme defines a committee key which is a commitment to a list of public keys. The committee key scheme can generate a succinct proof that a particular subset of the list of public keys signed a message. The proof can be verified using the committee key. Because of the way aggregatable signature scheme works (and, by extension, the committee key scheme), we need to specify the subset of signers; for this purpose we use a bitvector. More precisely, if the owner of the mth public key in the list of public keys signed the message, then the mth bit of this bitvector is 1 otherwise is 0. Using the committee key, the proof and the bitvector, a light client can verify that the corresponding subset of validators to whom the public keys belong (as per our use case) signed the message. Although the bitvector has length proportional to the number of validators, it is still orders of magnitude more succinct than giving all the public keys or signatures. Public keys or signatures are usually 100s of bits long and as a result this scheme reduces the amount of data required by a factor of 100 times or more. We could instantiate our committee key scheme using any universal SNARK scheme and suitable commitment scheme. However to avoid long prover times for large validator sets, we use custom SNARKs. This gives fast enough proof times for the use cases we consider: a prover with commodity hardware can generate these custom SNARK proofs in real time, i.e. as fast as the consensus generates instances of this problem.

Our Application: Accountable Light Client Scheme The application of our scheme is to light clients. To consider how our scheme might be useful and compare it to other approaches, we return to what a light client is used for. Light clients allow resource constrained devices such as browsers or phones to follow a decentralised consensus protocol. Another resource constrained device on which a light client is useful would be a blockchain -a light client verifier in this case (think smart contracts on Ethereum, for example) allow building trustless bridges protocols between blockchains [can cite rather than explain more here]. Currently, computation and storage costs on existing blockchains are much higher than those in a browser on a modern phone. If such a bridge is responsible for securing assets with high total value, then the corresponding light client system which defines such a light client verifier must be secure as well as efficient. Our light client system has the following properties which we explain below: It is accountable, has asynchronous safety and is incremental. Moreover, our solution is not only easily implementable on top of existing blockchains that use BLS signatures but also allows achieving accountability much more cheaply.

- Our light client system is accountable, i.e. if the light client verifier is misled and the transcript of its communication is given to the network then one can identify a large number (e.g. 1/3) of misbehaving consensus participants (i.e., validators in our specific case). Identifying misbehaving consensus participants is challenging in the light client system context when we want to send minimal data to the light client verifier. This is necessary for some proof of stake protocols, including Polkadot and Ethereum 2, whose security relies on identifying and punishing misbehaving consensus participants.

- Our light client system is asynchronously safe i.e. under the consensus' honesty assumptions, our light client verifier cannot be misled even if it has a restricted view of the network e.g. only connecting to one node, which may be malicious. This is because our light client system inherits the property of asynchronous safety from the Byzantine agreement protocol of the blockchain. Such light client systems would not be possible for consensus based on longest chains.

- Our light client system is incremental - i.e its succinct state is incrementally updated - is optimised to make these updates efficient, which is particularly relevant for the bridge application, as opposed to trying to optimise verifying consensus decisions from the blockchain genesis.

Technical challenges and contributions In order to define and implement our accountable light client systems and in order to design the custom SNARKs that support our efficiency results, we had to tackle some technical challenges and make additional contributions as summarised below.

Firstly, our custom SNARKs take inspiration from PLONK [11] in terms of the fundamentals of the proof system used and the design of circuits and gates. However, they also depart from PLONK in certain ways. Indeed, the type of NP relations for which PLONK can be used to implement SNARKs have their public inputs and witnesses interpreted as vectors of field elements. In order to design and prove the accountable light clients described in this work, we need SNARKs whose defining NP relations have polynomial commitments (i.e., our concrete instantiation for the committee key) as part of their public inputs as well. Hence, the original PLONK compiler does not suffice; we therefore extend it with a second step in which we show that under certain conditions (fulfilled by our light client scheme), the SNARKs obtained using the original PLONK compiler are also SNARKs for a mixed type of NP relation containing both vectors and polynomial commitments. The full details and proofs can be found in section 3.4 and we believe this compiler extension to be of independent interest, beyond our use case.

Secondly, besides the necessity of implementing SNARKs for NP relations containing both vectors of field elements and polynomial commitments, due to our specific application, we also require the NP relations we work with to have a well-defined subpredicate which is verified outside the SNARKs we design. In more detail, in our blockchain instantiation, any current validator set has to come to a consensus, among other things, on the next validator set which is represented, in turn, by a set of public keys. In fact, the current validator set computes and signs a pair of polynomial commitments[2] to the next set of validators' public keys; moreover, honest validators perform this task only if each of the public keys in the next set passes a subgroup check. This subgroup check is the subpredicate mentioned above and the security of our custom SNARKs relies on this property being fulfilled. Instead of making the subgroup check explicitly part of our custom SNARKs provers' circuits (and, thus, increasing the complexity of our SNARKs provers and verifiers), we prove security under the otherwise minimal assumption that each validator set has a given threshold/supermajority of honest participants which, in turn, ensures that the subpredicate in question verifies. This design decision makes our SNARKs more efficient, but it also means we have to extend the usual definition of NP relations to conditional NP relations, where in fact, one of the subpredicates that define the conditional relation is checked outside the SNARK or ensured due to a well-defined assumption. We introduce the general notion of conditional NP relation in section 2.4 and describe our concrete conditional NP relations in section 3.

Thirdly, in line with the two above technical challenges and the solutions we came up with, we also revisit the existing definitions related to SNARKs [12, 11] and we extend them by introducing an algorithm which we call *PartInput*. For our light client system use case, this allows us to separate the public input for the NP relations that define our custom SNARKs in two: a part that is computed by the current set of validators on the blockchain in question and the rest of the public input plus the corresponding SNARK proof are computed by a (possibly malicious) prover interacting with the light client verifier. Our newly introduced notion of hybrid model SNARK generalises this public input separation concept and its definition formalised in section 2.5 is used to the prove the security of our custom SNARKs in section 3.4.

1.1 Related Work

There exist two related approaches that describe a multi-epoch light client system with a verifier of constant size: Dfinity [10] uses a resharable BLS threshold signature and Mina [3] uses a ZK rollup of an entire blockchain including verifying its consensus. Both of these approaches have similar issues with accountability: if the light client verifier is misled, then the proof not only does not identify which of a particular set of validators are misbehaving, but we cannot say when this misbehaviour happened i.e. which validator set misbehaved. They also entail considerable computational or communication overhead.

1.1.1 Comparison to Celo's Plumo Protocol

Plumo [13] also tackles the problem we consider, i.e., that of proving validator set changes. In more detail, Plumo uses a Groth16 SNARK [12] to prove that enough validators signed a statement using BLS signatures from a set of the public keys. In Celo [4], the blockchain that designed and plans to use Plumo,

[2]These are exactly the polynomial commitments mentioned in the previous paragraph, which, for technical and efficiency reasons are part of each of our custom SNARKs' public inputs and the corresponding NP relation.

validators may change every epoch which is about a day long and the Plumo's SNARK iteratively proves 120 epochs worth of validator set changes. Since in Celo there are no more than 100 validators in a validator set at any one time, the respective public keys are used in plain as public input for Plumo's SNARK, as opposed to a succinct polynomial commitment in the case of our custom SNARKs. All of the above increase the size of Plumo's prover circuit. Since it is designed to help resource constrained light clients sync from scratch, it is not an impediment that the Plumo SNARK cannot be efficiently generated, i.e., in real time. In the case of a light client for bridges (i.e., the most resource constrained application), we expect it to be in sync at all times and, by design we care about only one validator set change at a time. Our slimed down and custom SNARK not only can be generated in real time, but, also due to the use of specialised commitments schemes for public keys, our validator sets can scale up to much larger numbers as well without impacting the efficiency of our system.

1.1.2 Why Not Use Threshold Signatures?

In a threshold scheme the signature does not depend on the set of signers. This feature is useful for implementing a common coin, but has the downside that there is no way to prove that a particular set of validator signed a message i.e. they are not accountable. The bigger downside of threshold signatures is that setting up the respective public key requires a complex multiparty protocol. While some threshold signatures also require interaction for signing messages, using schemes such as BLS can eliminate this need, but the setup process remains complex. Indeed, despite recent progress [14, 10, 15], it is still challenging to implement setup schemes for threshold signatures across a peer-to-peer network with large numbers of participants, which is what many blockchain related use cases require. Moreover, such a setup may need repeating whenever the signer set changes.

Lastly, several protocols [cite] have already implemented aggregatable BLS signatures and our committee key scheme can be used with those without altering the consensus layer. Indeed, few protocols [cite] appear to be using threshold signatures in practice, and it is easier to alter a protocol that uses individual signatures to using BLS signatures than to implement threshold signatures from scratch.

1.1.3 Commit-and-Prove and Related Approaches

Our custom SNARKs are an instance of the commit-and-prove paradigm [16, 17, 18] which, in turn, is a generalisation for zero-knowledge proofs/arguments in which the prover proves statements about values that are committed. In practice, commit-and-prove systems (for short, CP) can be used to compress a large data structure and then prove something about its content (e.g., polynomial commitments [19], vector commitments [20], accumulators [21]). CP schemes can also be used to decouple the publishing of commitments to some data from the proof generation: each of these actions may be performed by different parties or entities [22]. Finally, commitments can be used to make different proof systems interoperable [23, 24]. Our custom design SNARKs have properties from the first two categories, however we could not have simply re-used an existing argument system: by designing custom circuits and SNARKs, we ensured improved efficiency for our use cases.

Another paradigm related to commit-and-prove is called hash-and-prove [25]: for large data structures or simply data that is expensive to be handled directly by a computationally constrained verifier, one can hash that data and then create a (succinct) proof for some verifiable computation that uses the original, large, dataset. The committee key scheme notion that we define in this work has both similarities to but also differences with regard to this paradigm. The similarities are that, both the way we instantiate our committee key (i.e., using a polynomial commitment with a trusted universal setup) and the way we instantiate our aggregate public key, can be generalised as some form of (possibly deterministic) hash function. One difference is that the setup for the polynomial commitment is the same as that from which the proving and verification key for our committee key scheme are computed; thus our version of the hashes and the keys for the committee key scheme are definitely not independent as in the case of hash-and-commit [25]. Finally, built into our definition of committee key scheme and its security properties, we make use of a secure aggregatable signature scheme. This, in turn, allows us to design and prove the security properties of our accountable light client(s). In fact, to add some intuition to the fact that a committee key scheme is more than just a hash-and-prove instance, we mention that our committee

key scheme inherits an unforgeability property from its aggregatable scheme subcomponent. This is one property that as far as we are aware no hash-and-prove scheme has.

When proving the security of our arguments, we use an extension of some of the more commonly used SNARK definitions; we call this extension "a hybrid model SNARK". Our notion resembles the existing notion of SNARKs with online-offline verifiers as described in [25], where the verifier computation is split into two parts: during the offline phase some computation (possibly of commitments) happens; this computation takes some public inputs as parameters and, when not performed by the verifier, it may also be performed (in part) by the prover. The online phase is the main computation performed by the verifier. In the case of our hybrid model SNARKs, however, the input to the offline counterpart described above (which is what we call the *PartInput* algorithm) may even be the witness or a part of the witness for the respective relation. For our custom SNARKs, *PartInput* produces part of the public input used by the verifier; since for our use case, *PartInput* does handle a portion of the witness, this operation cannot be performed by the verifier for that relation. Moreover, in our instantiation, *PartInput* produces computationally binding commitment schemes that are opened by the prover. Both of these properties are not explicitly part of our general definition for hybrid model SNARKs, but they are crucial and explicitly assumed and used in proving the security for the result of our compiler's second step (see section 3.4).

2 Preliminaries

We assume all algorithms receive an implicit security parameter λ given in unary representation. We use interchangeably "efficient algorithm" or "PPT algorithm" to mean an algorithm that runs in uniform probabilistic polynomial time in the length of its input. Wherever necessary, before the run of all the algorithms and protocols, we assume the correct parameters for the curves, groups, parings, the group generators, etc. have been generated and shared with the corresponding parties.

We write $y = A(x; r)$ when algorithm A on input x and randomness r, outputs y. We write $y \leftarrow A(x)$ for the process of picking randomness r at random and setting $y = A(x; r)$. We also write $y \xleftarrow{\$} S$ for sampling y uniformly at random from the set S. We denote by $|S|$ the cardinality of set S. Unless otherwise stated, when we write that an event holds with some probability, we implicitly mean that the probability is computed over the randomness of all randomised algorithms involved. We say a function is negligible in λ and denote it by $negl(\lambda)$ if that function vanishes faster than the inverse of any polynomial in λ. We say that a function is overwhelming in λ if it has the form $1-$ some function negligible in λ. We also use the notation e.w.n.p. to mean except with negligible probability, or, equivalently, with overwhelming probability. We denote by $\mathsf{poly}(\lambda)$ an unspecified function which has a polynomial expression in λ. We generally use boldface font to denote vectors whose components we explicitly make use of in the text and we use italic font to denote the rest of the variables.

We work over finite fields of large characteristic. When we work with polynomials we denote by $\mathbb{F}_{<d}[X]$ the set of all polynomials of degree less than d over the field $\mathbb{F}$. For any integer $n \geq 1$, we denote by $[n]$ the set $\{1, \ldots, n\}$.

2.1 Pairings

If E is an elliptic curve defined over a prime field $\mathbb{F}_p$ of large characteristic p, we denote by $E(\mathbb{F}_p)$ the abelian group containing all the points $(x, y) \in (\mathbb{F}_p)^2$ that satisfy the elliptic curve equation along with the point at infinity. Let r be a large prime such that r divides $|E(\mathbb{F}_p)|$ and $gcd(p, r) = 1$. The *embedding degree of* E is the smallest integer k such that r divides $p^k - 1$. If k is small we say E is *pairing friendly*. We call $\mathbb{F}_p$ *the base field of* E and $\mathbb{F}_r$ (i.e., the prime field of characteristic r) *the scalar field of* E.

Pairing friendly curves are important to us in this work because they allow us to efficiently construct and instantiate aggregatable signatures and SNARKs. For a pairing friendly curve E as above, let $\mathbb{G}_1$,

$\mathbb{G}_2$ and $\mathbb{G}_T$ be appropriately chosen subgroups of order r in $E(\mathbb{F}_p)$, $E(\mathbb{F}_{p^l})$ (for some $l \leq k$)[3] and in the multiplicative group $\mathbb{F}^*_{p^k}$ of the extension field $\mathbb{F}_{p^k}$. The types of pairings we are interested in this work are mappings e which are secure [26, 27], efficiently computable, they are defined as $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ for which *bilinearity* (i.e., $e(a \cdot g_1, b \cdot g_2) = e(g_1, g_2)^{a \cdot b}$, $\forall a, b \in \mathbb{Z}_r$, $\forall g_1 \in \mathbb{G}_1$, $\forall g_2 \in \mathbb{G}_2$) and *non-degeneracy* (i.e., if g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, then $g_T = e(g_1, g_2)$ is a generator for $\mathbb{G}_T$) hold.

Our results in this work hold for a pair of pairing-friendly elliptic curves E_{inn} (*the inner curve*) and E_{out} (*the outer curve*) such that the base field of E_{inn} equals the scalar field of E_{out}. In line with the naming from [28], we call any pair of pairing friendly elliptic curves with such property a *pairing-friendly two-chain*. We denote by $\mathbb{F}$ the base field of E_{inn} and we call p its characteristic. We denote by r the characteristic of the scalar field of E_{inn}. We also denote by e_{inn} and by e_{out} the efficient, secure pairings over E_{inn} and E_{out}, respectively.

We further denote by $\mathbb{G}_{1,inn}$, $\mathbb{G}_{2,inn}$ and $\mathbb{G}_{T,inn}$ the two cyclic source groups and the cyclic target group for e_{inn} and $g_{1,inn}$, $g_{2,inn}$, $g_{T,inn}$ are uniformly random chosen generators of these three groups. Analogously, $\mathbb{G}_{1,out}$, $\mathbb{G}_{2,out}$ and $\mathbb{G}_{T,out}$ are the two cyclic source groups and the cyclic target group for e_{out} and $g_{1,out}$, $g_{2,out}$, $g_{T,out}$ are uniformly random chosen generators of these three groups. We consider $\mathbb{G}_{1,inn}$, $\mathbb{G}_{2,inn}$, $\mathbb{G}_{1,out}$, $\mathbb{G}_{2,out}$ with additive notation for their group operation and we consider $\mathbb{G}_{T,inn}$ and $\mathbb{G}_{T,out}$ with multiplicative notation. We additionally write $[x]_{1,inn} = x \cdot g_{1,inn}$, $[x]_{2,inn} = x \cdot g_{2,inn}$. We assume that the curves, groups and fields defined in the last two paragraphs have been generated using implicit security parameter λ.

Finally, we note that in our implementation we instantiate E_{inn} with BLS12-377 [28] and E_{out} with BW6-761 [29].

2.2 Secure Signature Aggregation

An aggregatable signature scheme is a signature scheme that compresses signatures issued using possibly different signing keys into one signature. Below we give the formal definition of an aggregatable signature scheme making explicit use of the proofs-of-possession (PoPs) key registration model as introduced and employed in [30]. This approach both maintains the general formal presentation clear and simple and allows for an easy transition to the aggregatable signature scheme instantiation used as part of our main (accountable light client system) construction. In particular, for our instantiation we use aggregatable BLS signatures that have a very efficient aggregation procedure by adding together keys and by multiplying together signatures, but they are vulnerable to rogue key attacks [30]; against these attacks one can protect using PoPs. This is in contrast to other aggregation procedures that do not require PoPs for security but incur a higher computational cost (e.g., due to the use of multi-scalar multiplication). Moreover, for our concrete use case of accountable light clients, our efficient and simple signature aggregation method results in a simple and more efficient custom argument scheme (i.e., SNARK), which, in turn, compensates for the cost of having to work with PoPs.

Definition 1. *(Aggregatable Signature Scheme) An aggregatable signature scheme consists of the following tuple of algorithms (AS.Setup, AS.GenerateKeypair, AS.VerifyPoP, AS.Sign, AS.AggregateKeys, AS.AggregateSignatures, AS.Verify) such that for implicit security parameter λ:*

- *$pp \leftarrow AS.Setup(aux_{AS})$: a setup algorithm that, given an auxiliary parameter aux_{AS}, outputs public protocol parameters pp.*
- *$((pk, \pi_{PoP}), sk) \leftarrow AS.GenerateKeypair(pp)$: a key pair generation algorithm that outputs a secret key sk, and the corresponding public key pk together with a proof of possession π_{PoP} for the secret key.*

[3] $E(\mathbb{F}_{p^l})$ is the group of all points $(x, y) \in (\mathbb{F}_{p^l})^2$ that satisfy the elliptic curve equation of E along with the point at infinity.

- $0/1 \leftarrow AS.VerifyPoP(pp, pk, \pi_{PoP})$*: a public key verification algorithm that, given a public key* pk *and a proof of possession* π_{PoP}*, outputs* 1 *if* π_{PoP} *is valid for* pk *and* 0 *otherwise.*
- $\sigma \leftarrow AS.Sign(pp, sk, m)$*: a signing algorithm that, given a secret key* sk *and a message* $m \in \{0,1\}^*$*, returns a signature* σ*.*
- $apk \leftarrow AS.AggregateKeys(pp, (pk_i)_{i=1}^u)$*: a public key aggregation algorithm that, given a vector of public keys* $(pk_i)_{i=1}^u$*, returns an aggregate public key* apk*.*
- $asig \leftarrow AS.AggregateSignatures(pp, (\sigma_i)_{i=1}^u)$*: a signature aggregation algorithm that, given a vector of signatures* $(\sigma_i)_{i=1}^u$*, returns an aggregate signature* $asig$*.*
- $0/1 \leftarrow AS.Verify(pp, apk, m, asig)$*: a signature verification algorithm that, given an aggregate public key* apk*, a message* $m \in \{0,1\}^*$*, and an aggregate signature* σ*, returns 1 or 0 to indicate if the signature is valid.*

We say $(AS.Setup, AS.GenerateKeypair, AS.VerifyPoP, AS.Sign, AS.AggregateKeys, AS.AggregateSignatures, AS.Verify)$ *is an aggregatable signature scheme if it satisfies* perfect completeness*,* perfect completeness for aggregation *and* unforgeability *as defined below.*

Perfect Completeness An aggregatable signature scheme ($AS.Setup$, $AS.GenerateKeypair$, $AS.VerifyPoP$, $AS.Sign$, $AS.AggregateKeys$, $AS.AggregateSignatures$, $AS.Verify$) has perfect completeness if for any message $m \in \{0,1\}^*$ and any $u \in \mathbb{N}$ it holds that:

$$\begin{aligned} Pr[AS.Verify(pp, apk, m, asig) = 1 \ \wedge \ \forall i \in [u] \ AS.VerifyPoP(pp, pk_i, \pi_{PoP,i}) = 1 \mid \\ pp \leftarrow AS.Setup(aux_{AS}), \\ ((pk_i, \pi_{PoP,i}), sk_i) \leftarrow AS.GenerateKeypair(pp), \ i = 1, \ldots, u \\ apk \leftarrow AggregateKeys(pp, (pk_i)_{i=1}^u), \\ \sigma_i \leftarrow AS.Sign(pp, sk_i, m), \ i = 1, \ldots, u, \\ asig \leftarrow AS.AggregateSignatures(pp, (\sigma_i)_{i=1}^u)] = 1. \end{aligned}$$

We note that an aggregatable signature scheme with perfect completeness implies the underlying signature scheme has perfect completeness.

Perfect Completeness for Aggregation An aggregatable signature scheme ($AS.Setup$, $AS.GenerateKeypair$, $AS.VerifyPoP$, $AS.Sign$, $AS.AggregateKeys$, $AS.AggregateSignatures$, $AS.Verify$) has perfect completeness for aggregation if, for every adversary $\mathcal{A}$

$$\begin{aligned} Pr[AS.Verify(pp, apk, m, asig) = 1 \mid pp \leftarrow AS.Setup(aux_{AS}), \\ ((pk_i)_{i=1}^u, m, (\sigma_i)_{i=1}^u) \leftarrow \mathcal{A}(pp), \\ \forall i \in [u], AS.Verify(pp, pk_i, m, \sigma_i) = 1, \\ apk \leftarrow AS.AggregateKeys(pp, (pk_i)_{i=1}^u), \\ asig \leftarrow AS.AggregateSignatures(pp, (\sigma_i)_{i=1}^u)] = 1. \end{aligned}$$

Unforgeable Aggregatable Signature For an aggregatable signature scheme ($AS.Setup$, $AS.GenerateKeypair$, $AS.VerifyPoP$, $AS.Sign$, $AS.AggregateKeys$, $AS.AggregateSignatures$, $AS.Verify$) the advantage of an adversary against unforgeability is defined by

$$Adv_{\mathcal{A}}^{forge}(\lambda) = Pr[Game_{\mathcal{A}}^{forge}(\lambda) = 1]$$

where

$$
\begin{aligned}
&Game_{\mathcal{A}}^{forge}(\lambda):\\
&pp \leftarrow AS.Setup(aux_{AS})\\
&((pk^*, \pi^*_{PoP}), sk^*) \leftarrow AS.GenerateKeypair(pp)\\
&Q \leftarrow \emptyset\\
&((pk_i, \pi_{PoP,i})_{i=1}^{u}, m, asig) \leftarrow \mathcal{A}^{OSign}(pp, (pk^*, \pi^*_{PoP}))\\
&If\ pk^* \notin \{pk_i\}_{i=1}^{u} \vee m \in Q,\ then\ return\ 0\\
&For\ i \in [u]\\
&\quad If\ AS.VerifyPoP(pp, pk_i, \pi_{PoP,i}) = 0\ return\ 0\\
&apk \leftarrow AS.AggregateKeys(pp, (pk_i)_{i=1}^{u})\\
&Return\ AS.Verify(pp, apk, m, asig)
\end{aligned}
$$

and

$$
\begin{aligned}
&OSign(m_j):\\
&\sigma_j \leftarrow AS.Sign(pp, sk^*, m_j)\\
&Q \leftarrow Q \cup \{m_j\}\\
&Return\ \sigma_j
\end{aligned}
$$

and $\mathcal{A}^{OSign}$ denotes the adversary $\mathcal{A}$ with access to oracle $OSign$.

We say an aggregatable signature scheme is unforgeable if for all efficient adversaries $\mathcal{A}$ it holds that $Adv_{\mathcal{A}}^{forge}(\lambda) \leq negl(\lambda)$.

2.2.1 An Aggregatable Signature Instantiation

In the following, we instantiate the aggregatable signature definition given above with a scheme inspired by the BLS signature scheme [8] and its follow-up variants [30, 9].

Instantiation 2. *(Aggregatable Signatures) In our implementation we call aggregatable signatures the following instantiation of aggregatable signatures definition. Note that in our implementation we instantiate E_{inn} with BLS12-377 [28].*

- $(\mathbb{G}_{1,inn}, g_{1,inn}, \mathbb{G}_{2,inn}, g_{2,inn}, \mathbb{G}_{T,inn}, e_{inn}, H_{inn}, H_{PoP}) \subset pp \leftarrow AS.Setup(aux_{AS})$, *where $\mathbb{G}_{1,inn}$, $g_{1,inn}$, $\mathbb{G}_{2,inn}$, $g_{2,inn}$, $\mathbb{G}_{T,inn}$, e_{inn} were defined in section 2.1 and $H_{inn} : \{0,1\}^* \to \mathbb{G}_{2,inn}$ and $H_{PoP} : \{0,1\}^* \to \mathbb{G}_{2,inn}$ are two hash functions. The auxiliary parameter aux_{AS} is such that there exists $N \in \mathbb{N}$, N is the first component of the vector aux_{AS} and there exists a subgroup of size at least N in the multiplicative group of $\mathbb{F}$, where $\mathbb{F}$ is the base field of E_{inn}, but also the size of the subgroup $\in O(N)$.*

- $(pk, sk, \pi_{inn}) \leftarrow AS.GenerateKeypair(pp)$, *where $sk \xleftarrow{\$} \mathbb{Z}_r^*$ and $pk = sk \cdot g_{1,inn} \in \mathbb{G}_{1,inn}$ and $\pi_{inn} \leftarrow sk \cdot H_{PoP}(pk)$ and r was defined in section 2.1 as the characteristic of the scalar field of E_{inn}.*

- $0/1 \leftarrow AS.VerifyPoP(pp, pk, \pi_{inn})$, *where $AS.VerifyPoP$ outputs 1 if*

$$e_{inn}(g_{1,inn}, \pi_{inn}) = e_{inn}(pk, H_{PoP}(pk))$$

  holds and 0 otherwise. Note that implicitly, as part of running $AS.VerifyPoP$, one checks that $pk \in \mathbb{G}_{1,inn}$ also holds.

- $\sigma \leftarrow AS.Sign(pp, sk, m)$*: where $\sigma = sk \cdot H_{inn}(m) \in \mathbb{G}_{2,inn}$.*

- $apk \leftarrow AS.AggregateKeys(pp, (pk_i)_{i=1}^u)$, *where* $apk = \sum_{i=1}^u pk_i$.
- $asig \leftarrow AS.AggregateSignatures(pp, (\sigma_i)_{i=1}^u)$, *where* $asig = \sum_{i=1}^u \sigma_i$.
- $0/1 \leftarrow AS.Verify(pp, apk, m, asig)$, *where* $AS.Verify$ *outputs* 1 *if* $e_{inn}(apk, H_{inn}(m)) = e_{inn}(g_{1,inn}, asig)$ *and else outputs* 0.

2.3 Committee Key Scheme for Aggregatable Signatures

Bellow we introduce the notion of committee key scheme for aggregatable signatures. This generalises the notion of aggregatable signature scheme. We will use the notion of committee key scheme and its instantiation presented in section 3.6 in order to design, instantiate and prove our accountable light client schemes in section 4.

Definition 3. *(Committee Key Scheme for Aggregatable Signatures) Let AS be an aggregatable signature scheme that fulfils definition 1. A committee key scheme for aggregatable signatures consists of the following tuple of algorithms (CKS.Setup, CKS.GenerateCommitteeKey, CKS.Prove, CKS.Verify) such that for implicit security parameter* λ*:*

- $(pp, rs_{vk}, rs_{pk}) \leftarrow CKS.Setup(v)$*: a setup algorithm that, given an upper bound* $v \in \mathbb{N}$, $v = \mathsf{poly}(\lambda)$ *outputs some public parameters pp and proving and verification keys* rs_{pk} *and* rs_{vk}*, respectively, where* $pp \leftarrow AS.Setup(aux_{AS})$*, for some* aux_{AS} *chosen by the aggregated signature AS.*
- $ck \leftarrow CKS.GenerateCommitteeKey(rs_{pk}, (pk_i)_{i=1}^u)$*: a committee key generation algorithm that, given a proving key* rs_{pk} *and a list of public keys, outputs a committee key ck, where* $u \leq v$.
- $\pi \leftarrow CKS.Prove(rs_{pk}, ck, (pk_i)_{i=1}^u, (bit_i)_{i=1}^u)$*: a proving algorithm that, given a proving key* rs_{pk}*, a committee key ck, a list of public keys and a bitvector* $(bit_i)_{i=1}^u \in \{0,1\}^u$*, outputs a proof* π*, where* $u \leq v$.
- $0/1 \leftarrow CKS.Verify(pp, rs_{vk}, ck, m, asig, \pi, \mathbf{bitvector})$*: a verification algorithm that, given public parameters pp, a verification key* rs_{vk}*, a committee key ck, a message m, a signature asig, a proof* π *and a vector* $\mathbf{bitvector} \in \{0,1\}^*$*, outputs* 1 *if the verification succeeds and* 0 *otherwise.*

We say (CKS.Setup, CKS.GenerateCommitteeKey, CKS.Prove, CKS.Verify) is a committee key scheme for aggregatable signatures if it satisfies perfect completeness *and* soundness *as defined below.*

Perfect Completeness A committee key scheme for aggregatable signatures (*CKS.Setup*, *CKS.GenerateCommitteeKey*, *CKS.Prove*, *CKS.Verify*) has perfect completeness if for any message $m \in \{0,1\}^*$, for any vector of public keys $(pk_i)_{i=1}^u$ generated using $AS.GenerateKeypair(pp)$, for any bitmask $(bit_i)_{i=1}^u \in \{0,1\}^u$, for any aggregated signature *asig*, it holds that:

$$\begin{aligned} Pr[AS.Verify(pp, apk, m, asig) = 1 \implies & CKS.Verify(pp, rs_{vk}, ck, m, asig, \pi, (bit_i)_{i=1}^u) = 1 | \\ & (pp, rs_{vk}, rs_{pk}) \leftarrow CKS.Setup(v), \\ & ck \leftarrow CKS.GenerateCommitteeKey(rs_{pk}, (pk_i)_{i=1}^u)), \\ & \pi \leftarrow CKS.Prove(rs_{pk}, ck, (pk_i)_{i=1}^u, (bit_i)_{i=1}^u), \\ & apk \leftarrow AS.AggregateKeys(pp, (pk_i)_{i:bit_i=1})] = 1 \end{aligned}$$

Soundness A committee key scheme for aggregatable signatures (*CKS.Setup*, *CKS.GenerateCommitteeKey*, *CKS.Prove*, *CKS.Verify*) has soundness if for every efficient adversary $\mathcal{A}$ it holds that:

$$\begin{aligned} Pr[CKS.Verify(pp, rs_{vk}, ck, m, asig, \pi, (bit_i)_{i=1}^u) = 1 \implies & AS.Verify(pp, apk, m, asig) = 1 | \\ & (pp, rs_{vk}, rs_{pk}) \leftarrow CKS.Setup(v), \\ & (pk_i)_{i=1}^u, (bit_i)_{i=1}^u, asig, \pi, m \leftarrow \mathcal{A}(pp, rs_{vk}, rs_{pk}), \\ & ck \leftarrow CKS.GenerateCommitteeKey(rs_{pk}, (pk_i)_{i=1}^u), \\ & apk \leftarrow AS.AggregateKeys(pp, (pk_i)_{i:bit_i=1})] = 1 - negl(\lambda) \end{aligned}$$

Next, we define an additional security property for a committee key scheme for aggregatable signatures, namely unforgeability.

Unforgeability For a committee key scheme for aggregatable signatures ($CKS.Setup$, $CKS.GenerateCommitteeKey$, $CKS.Prove$, $CKS.Verify$) the advantage of an adversary $\mathcal{A}$ against unforgeability is defined by $Adv_{\mathcal{A}}^{forgecomkey}(\lambda) = \Pr[Game_{\mathcal{A}}^{forgecomkey}(\lambda) = 1]$, where

$Game_{\mathcal{A}}^{forgecomkey}(\lambda)$:
$(pp, rs_{vk}, rs_{pk}) \leftarrow CKS.Setup(v)$
$((pk^*, \pi_{PoP}^*), sk^*) \leftarrow AS.GenerateKeypair(pp)$
$Q \leftarrow \emptyset$
$((pk_i, \pi_{PoP,i})_{i=1}^u, (bit_i)_{i=1}^u, asig, \pi, m) \leftarrow \mathcal{A}^{OSign}(pp, rs_{vk}, rs_{pk}, (pk^*, \pi_{PoP}^*))$
If $(\forall i : pk^* \neq pk_i \wedge bit_i = 0) \vee m \in Q$, *then return* 0
For $i \in [u]$
    If $AS.VerifyPoP(pp, pk_i, \pi_{PoP,i}) = 0$ *return* 0
$ck \leftarrow CKS.GenerateCommitteKey(rs_{pk}, (pk_i)_{i=1}^u)$
Return $CKS.Verify(pp, rs_{vk}, ck, m, asig, \pi, (bit_i)_{i=1}^u)$

We say a committee key scheme for aggregatable signatures is unforgeable if for all efficient adversaries $\mathcal{A}$ it holds that $Adv_{\mathcal{A}}^{forgecomkey}(\lambda) \leq negl(\lambda)$.

Corollary 4. *Let AS be an aggregatable signature scheme that fulfils definition 1. If CKS is a committee key scheme for aggregatable signatures that fulfils definition 3, then CKS is unforgeable, as defined above.*

Proof. Assume by contradiction there exists an efficient adversary $\mathcal{A}$ such that $Adv_{\mathcal{A}}^{forgecomkey}(\lambda)$ is non-negligible. Using $\mathcal{A}$ and the soundness property of a committee key scheme, one can construct in a straightforward manner an efficient adversary $\mathcal{A}'$ such that

$$Adv_{\mathcal{A}'}^{forge}(\lambda) \geq Adv_{\mathcal{A}}^{forgecomkey}(\lambda) - negl(\lambda).$$

This, in turn, implies that $Adv_{\mathcal{A}'}^{forge}$ is non-negligible which contradicts the unforgeability property of aggregatable signature scheme AS. Thus, our assumption is false and our statement holds. □

2.4 Conditional NP Relations

By $\mathcal{R} = \{(x; w) : p(x, w) = 1\}$ we denote the binary relation such that (x, w) fulfil predicate $p(x, w) = 1$. We say $\mathcal{R}$ is an NP relation if predicate p can be checked in polynomial time in the length of both inputs x and w and $\mathcal{L}(\mathcal{R}) = \{x \mid \exists w \ s.t. \ (x, w) \in \mathcal{R}\}$ is an NP language w.r.t. predicate p. In such a case we call x an *instance* and w a *witness*.

In order to model a specific property of our NP relations, we introduce further notation which we call *conditional NP relation*, we denote it by

$$\mathcal{R}^c = \{(x; w) : (p_1(x, w) = 1 \mid c(x, w) = 1) \ \wedge \ p_2(x, w) = 1\}$$

and we interpret as the NP relation containing the pairs of inputs and witnesses (x, w) such that $c(x, w) = 1$, $p_1(x, w) = 1$ and $p_2(x, w) = 1$ hold. However, in order to prove that $(x, w) \in \mathcal{R}^c$ we assume/take it as a given that $c(x, w) = 1$ and we are left to prove only that $p_1(x, w) = 1$ and $p_2(x, w) = 1$ hold.

We explicitly include in the definition of any NP relation $\mathcal{R}$ or $\mathcal{R}^c$ the corresponding domain for each type of public input. The interpretation of such domains is that each type of public input is parsed by the honest parties (e.g., a SNARK verifier for an NP relation $\mathcal{R}$ or $\mathcal{R}^c$) as per the definition of the respective domain, without additional checks. We assume that all our relations have been generated using implicit security parameter λ. Finally, if not stated explicitly, when we make a statement about an NP relation we implicitly mean the statement is about a conditional relation $\mathcal{R}^c$, where c may be the predicate that always outputs 1.

2.5 SNARKs

All three SNARKs we design in this work have access to a *structured reference string* (srs) of the form $(\{[\tau^i]_1\}_{i=0}^{d}, \{[\tau^i]_2\}_{i=0}^{1})$ where τ is a random (and allegedly secret) value in $\mathbb{F}$ and d is bounded by a polynomial in λ. Such an srs is *universal* and *updatable* [31] and, as long as at least one of the participants that took part in the MPC generating the srs was honest, the srs cannot be used by any coalition of other MPC participants to prove false statements with more than a negligible probability of success [31, 32].

Our SNARKs are secure in the *algebraic group model* (AGM) [33]. If $\mathbb{G}$ is a cyclic group of prime order p, then, informally, we call an algorithm $\mathcal{A}$ *algebraic* if it fulfils the following requirement: whenever $\mathcal{A}$ outputs a group element $g \in \mathbb{G}$, it also outputs a representation $\mathbf{a} = (a_1, ..., a_t) \in \mathbb{Z}_p^t$ such that $g = \sum_{i=1}^{t} a_i \cdot B_i$ where $(B_1, \ldots, B_t)$ are all the $\mathbb{G}$ group elements that were given to $\mathcal{A}$ during its execution so far. The AGM lies in between the *generic group model* (GGM) [34, 35] and the standard model and, lately, it has been the preferred model for proving security for the most efficient SNARKs (e.g., PLONK [11], Marlin [36] or Groth16 [12] with its proof in the AGM model presented in [33, 37]).

In the following, we introduce a generalisation of the usual SNARK definition which we call a *hybrid model SNARK*. As mentioned in the introduction, this is inspired by the notion of online-offline SNARKs [25], however, for our use case we need to further refine it as describe below:

Definition 5. *(Hybrid Model SNARK) A hybrid model* succinct non-interactive argument of knowledge for relation $\mathcal{R}$ *is a tuple of PPT algorithms* $(SNARK.Setup, SNARK.KeyGen, SNARK.Prove, SNARK.Verify, SNARK.PartInputs)$ *such that for implicit security parameter* λ*:*

- $srs \leftarrow SNARK.Setup(aux_{SNARK})$*: a setup algorithm that on input auxiliary parameter* aux_{SNARK} *from some domain* $\mathcal{D}$ *outputs a universal structured reference string tuple* srs*,*
- $(srs_{pk}, srs_{vk}) \leftarrow SNARK.KeyGen(srs, \mathcal{R})$*: a key generation algorithm that on input a universal structured reference string* srs *and an NP relation* $\mathcal{R}$ *outputs a* proving key *and a* verification key *pair* (srs_{pk}, srs_{vk})*,*
- $\pi \leftarrow SNARK.Prove(srs_{pk}, (x, w), \mathcal{R})$*: a proof generation algorithm that on input a proving key* srs_{pk} *and a pair* $(x, w) \in \mathcal{R}$ *outputs* proof π*,*
- $0/1 \leftarrow SNARK.Verify(srs_{vk}, x, \pi, \mathcal{R})$*: a proof verification algorithm that on input a verification key* srs_{vk}*, an instance* x *and a proof* π *outputs a bit that signals acceptance (if output is* 1*) or rejection (if output is* 0*)*
- $(x_1, state_2) \leftarrow SNARK.PartInputs(srs, state_1, \mathcal{R})$*: a deterministic public inputs generation algorithm that takes as input a universal structured reference string* srs*, an NP relation* $\mathcal{R}$ *and some state* $state_1$ *and outputs some updated state* $state_2$ *and some partial public input* x_1*,*

and satisfies completeness, knowledge soundness with respect to $SNARK.PartInputs$ *and succinctness as defined below:*
Perfect Completeness *holds if an honest prover will always convince an honest verifier: for all* $(x, w) \in \mathcal{R}$ *and for all* $aux_{SNARK} \in \mathcal{D}$

$$Pr[SNARK.Verify(srs_{vk}, x, \pi, \mathcal{R}) = 1 \mid \begin{array}{l} srs \leftarrow SNARK.Setup(aux_{SNARK}), \\ (srs_{pk}, srs_{vk}) \leftarrow SNARK.KeyGen(srs, \mathcal{R}), \\ \pi \leftarrow SNARK.Prove(srs_{pk}, (x, w), \mathcal{R}) \end{array}] = 1.$$

Notation *In the following, we denote by* $State_{\mathcal{R}}$ *the set of all states* $state_1$ *such that given some relation* $\mathcal{R}$ *and any possible* srs*, for any output* x_1 *of* $SNARK.PartInputs(srs, \mathcal{R}, state_1)$ *with* $state_1 \in State_{\mathcal{R}}$*, we have that there exists* x_2 *and* w *with* $(x = (x_1, x_2), w) \in \mathcal{R}$*; we further make the assumption that* $State_{\mathcal{R}} \neq \emptyset$*.*

Knowledge-soundness with respect to *SNARK.PartInputs holds if there exists a PPT extractor* $\mathcal{E}$ *such that for all PPT adversaries* $\mathcal{A}$*, for all* $aux_{SNARK} \in \mathcal{D}$ *and for all* $state_1 \in State_{\mathcal{R}}$

$$Pr[(x = (x_1, x_2), w) \in \mathcal{R} \wedge 1 \leftarrow SNARK.Verify(srs_{vk}, x = (x_1, x_2), \pi, \mathcal{R}) \mid$$
$$srs \leftarrow SNARK.Setup(aux_{SNARK}), (srs_{pk}, srs_{vk}) \leftarrow SNARK.KeyGen(srs, \mathcal{R}),$$
$$(x_1, state_2) \leftarrow SNARK.PartInput(srs, state_1, \mathcal{R}), (x_2, \pi) \leftarrow \mathcal{A}(srs, state_2, \mathcal{R}), w \leftarrow \mathcal{E}^{\mathcal{A}}(srs, state_2, \mathcal{R})]$$

is overwhelming in λ*, where by* $\mathcal{E}^{\mathcal{A}}$ *we denote the extractor* $\mathcal{E}$ *that has access to all of* $\mathcal{A}$*'s messages during the protocol with the honest verifier.*
Succinctness *holds if the size of the proof* π *is* $\mathsf{poly}(\lambda)$ *and SNARK.Verify runs in time* $\mathsf{poly}(\lambda + |x|)$*.*

Firstly, note that if one chooses x_1, $state_1$ and $state_2$ to be the empty strings in the definition of *SNARK.PartInput* and in relation to the knowledge soundness property, one obtains a more standard SNARK definition. Secondly, $\mathcal{R}$ is not a component of the vector aux_{SNARK} so even if *SNARK.Setup* has aux_{SNARK} as parameter, it is universal, i.e., it can be used to derive proving and verification keys for circuits of any size up to a polynomial in the security parameter λ, independently of any specific NP relation. Moreover, for the SNARKs we design, the size of the key used by the honest verifier is much smaller than the size of the honest prover's key. We have made the separation clear between the two keys to be able to better capture this special case; however, a potential adversarial prover has access to the complete *srs* key. Thirdly, as mentioned the SNARKs that we design in this work are secure in the *AGM* model. This means that we limit our adversaries to *AGM* adversaries only and by $\mathcal{E}^{\mathcal{A}}$ we denote the extractor $\mathcal{E}$ that has access to all of $\mathcal{A}$'s messages during the protocol with the honest verifier: the messages include the coefficients of the linear combinations of group elements used by the *AGM* adversary at any step in order to output new group elements at the next step in the protocol. Moreover, the auxiliary input (i.e., $state_2$) is required to be drawn from a "benign distribution" or else extraction may be impossible [38, 39]. Finally, in the SNARK definition above we did not include the notion of zero-knowledge since it is not required in the rest of the paper.

2.6 Ranged Polynomial Protocols and Polynomial Commitments

In order to prove the security of the SNARKs designed in this work we use a SNARK compiler inspired by the one provided in lemma 4.7 from PLONK [11]. In more detail, for each of our three conditional NP relations we describe a ranged polynomial protocol and then we use our compiler to obtain three SNARKs secure in the AGM. We remind the definition of ranged polynomial protocols in appendix A. Moreover, we also make use of KZG polynomial commitments [19], in particular their batched version and their security definitions as described in section 3 from PLONK. For brevity, and since we do not make any alterations to the definition of batched KZG commitments, we do not repeat it in this initial version of our work but invite the reader to review them, if necessary, by following the reference provided.

2.7 Lagrange Bases

In order to design the SNARKs presented in this work, it is more convenient to represent the polynomials we work with over the Lagrange base rather than the monomial base. Formally, for the finite field $\mathbb{F}$ defined in section 2.1 we denote by H a subgroup of the multiplicative group of $\mathbb{F}$ such that $n = |H|$ is a large power of 2. Let ω be an n-th root of unity in $\mathbb{F}$ such that ω is a generator of H. Then, we call the following polynomial base $\{L_i(X)\}_{0 \leq i \leq n-1}$ a Lagrange base, where $\forall i, 0 \leq i \leq n-1$, $L_i(X)$ is the unique polynomial in $\mathbb{F}_{<n}[X]$ such that $L_i(\omega^i) = 1$ and $L_i(\omega^j) = 0, \forall j \neq i$.

Independent of the notion of Lagrange bases, but related to n we define block also a power of 2 such that $\mathsf{block} < n$. We use block when defining one of our conditional NP relations in section 3. In the following we assume $n = \mathsf{poly}(\lambda)$ and $\mathsf{block} = \Theta(\lambda)$ and $|\mathbb{F}| = 2^{\Theta(\lambda)}$.

3 Public Keys Aggregation Proofs using Custom SNARKS

In the following, we construct three related SNARKS, each of them allowing a prover to convince an efficient verifier that an alleged aggregated public key has indeed been computed correctly as an aggregate of a vector of public keys for which two succinct commitments (one to the vector of x affine coordinates and the other to the vector of y affine coordinates, respectively) are publicly known. The differences between the three constructions stem from whether a *bitmask* (also called a *bitvector*) with one bit associated to each public key (necessary to signal the inclusion or omission of the respective public key w.r.t. the aggregate key) is part of the verifier's public input or is part of the witness. For the former case, we describe, in fact, two distinct SNARKs: a basic accountable SNARK (the bitmask is represented as a sequence of $\{0, 1\}$ field elements) and a packed accountable SNARK (the bitmask is partitioned into equal blocks of consecutive binary bits, and, in turn, each block is represented as a field element). For the latter case, we describe a counting SNARK. Each of our three SNARKs implements a conditional NP relation bearing the same name as the SNARK it implements. Note that the names "basic accountable" (for short, "basic"), "packed accountable" (for short, "packed") and "counting" do not refer to the security of the respective SNARK but they summarise properties of the underlying sets of constraints that define the SNARKs, and, hence their use case. In particular, we use the basic accountable and the packed accountable SNARKs for building accountable light client systems and we use the counting SNARK for building non-accountable light client systems.

In order to compile our desired SNARKs we proceed as follows:

- We start by defining three conditional NP relations based only on vectors. These relations capture the specific constraints we are interested in. We denote these NP relations by $\mathcal{R}^{incl}_{\mathsf{ba}}$ (i.e., basic accountable), $\mathcal{R}^{incl}_{\mathsf{pa}}$ (packed accountable) and $\mathcal{R}^{incl}_{\mathsf{c}}$ (counting). (See sections 3.1, 3.2 and 3.3, respectively, for full details.)
- We design three ranged polynomial protocols for the above three relations. (Again, see sections 3.1, 3.2 and 3.3, respectively, for full details.). The definition of ranged polynomial protocols originates in [11] and, for convenience, we remind it to the reader in appendix A.
- We define and use a two-steps PLONK-based compiler that allows us to compile the three ranged polynomial protocols into the desired SNARKs for a novel type of conditional NP relations which include trusted inputs and, in particular, trusted polynomial commitments as part of the public inputs. In more detail, we compile three SNARKs for three mixed polynomial commitments and vector based relations which we denote by $\mathcal{R}^{incl}_{\mathsf{ba},com}$, $\mathcal{R}^{incl}_{\mathsf{pa},com}$ and $\mathcal{R}^{incl}_{\mathsf{c},com}$, respectively. These are the direct counterparts of pure vector based relations $\mathcal{R}^{incl}_{\mathsf{ba}}$, $\mathcal{R}^{incl}_{\mathsf{pa}}$ and $\mathcal{R}^{incl}_{\mathsf{c}}$. (See section 3.4 for full details. For completeness, we also include in appendix B the full rolled-out SNARK implementing $\mathcal{R}^{incl}_{\mathsf{pa},com}$.)
- We include a detailed comparison between the original PLONK and our SNARKs. (See section 3.5.)
- We conclude this section with an instantiation for committee key scheme for aggregatable signatures which uses, in turn, our SNARKS compiled in section 3.4 and our instantiation for BLS aggregatable signatures from section 2.2.1. (See section 3.6 for full details.)

In more detail, as motivated in the introduction and in section 2.1, we define all our conditional NP relations for our three SNARKs over $\mathbb{F}$ which is the base field of the curve E_{inn}. Moreover, our corresponding SNARKs provers' circuits are naturally defined as well over $\mathbb{F}$ as the scalar field of E_{out}. In particular, the vector of public keys, which is part of the public input for all of our three relations, and is denoted by $\mathbf{pk} = (pk_0, \ldots, pk_{n-2})$, is a vector of pairs with each component in $\mathbb{F}$. Note that this vector has size $n-1$ where n has been defined in section 2.7. For the basic accountable and the counting conditional NP relations, we denote the n components bitmask by $\mathbf{bit} = (bit_0, \ldots, bit_{n-1})$ (meaning that each component belongs to the set $\{0, 1\} \subset \mathbb{F}$), while the packed accountable conditional NP relation is defined using the *compacted bitmask* $\mathbf{b'} = (b'_0, \ldots, b'_{\frac{n}{\mathsf{block}}-1})$ of $\frac{n}{\mathsf{block}}$ field elements, each of which is block binary bits long

(block has been defined in section 2.7). Intuitively, each of the binary bits in the bit representation of these field elements signals the inclusion (or exclusion) of the index-wise corresponding public keys into the aggregated public key apk. Note that, in fact, the last bit of field element $b'_{\frac{n}{\mathsf{block}}-1}$ as well as the n-th component bit_{n-1} do not correspond to any public key, but, as will become clear in the following, they have been included for easier design of constraints.

Notation-wise, we denote by H the multiplicative subgroup of $\mathbb{F}$ generated by ω as defined in section 2.7. We additionally denote by $incl(a_0, \ldots, a_{n-2})$ the inclusion predicate that checks if $(a_0, \ldots, a_{n-2}) \in \mathbb{G}^{n-1}_{1,inn}$. Moreover let $h = (h_x, h_y)$ be some fixed, publicly known element in $E_{inn} \setminus \mathbb{G}_{1,inn}$. (See full version of this work for how to handle the special case $E_{inn} = \mathbb{G}_{1,inn}$.) We denote by (a_x, a_y) the affine representation in x and y coordinates of $a \in E_{inn}$ and by $\oplus$ the point addition in affine coordinates on the elliptic curve E_{inn}. We denote by $[s]P$ the scalar multiplication by scalar $s \in \mathbb{F}$ of point $P \in E_{inn}$. We denote by $\mathbb{B} = \{0, 1\} \subset \mathbb{F}$.

Finally, as mentioned in section 2.4, the interpretation of adding explicit domains to public inputs in the definition of conditional NP relations is that the honest parties (in our case, both the polynomial protocol verifiers and the SNARKs verifiers as defined in this section below) parse the public inputs according to the specified domains without any further checks. Any checks or computations that the honest parties perform regarding the public inputs are explicitly described as part of the protocols followed by the honest parties.

3.1 Basic Accountable Ranged Polynomial Protocol

We start by describing our conditional basic accountable relation $\mathcal{R}^{incl}_{\mathsf{ba}}$ and the corresponding H-ranged polynomial protocol $\mathscr{P}_{\mathsf{ba}}$. Both n and the domains used in the explicit definitions of our conditional NP relations depend implicitly on the security parameter λ, hence $\mathcal{R}^{incl}_{\mathsf{ba}}$ as well implicitly depends on λ. However, for brevity, here and in the rest of the paper we choose to omit the security parameter λ whenever we refer to any of the conditional NP relations for which we build our SNARKs

Conditional Basic Accountable Relation $\mathcal{R}^{incl}_{\mathsf{ba}}$

$$\mathcal{R}^{incl}_{\mathsf{ba}} = \{(\mathbf{pk} \in (\mathbb{F}^2)^{n-1}, \mathbf{bit} \in \mathbb{B}^n, apk \in \mathbb{F}^2; \_) : apk = \sum_{i=0}^{n-2} [bit_i] \cdot pk_i \mid \mathbf{pk} \in \mathbb{G}^{n-1}_{1,inn}\}$$

where $\mathbf{pk} = (pk_0, \ldots, pk_{n-2})$ and $\mathbf{bit} = (bit_0, \ldots, bit_{n-1})$. Throughout this section we are going to use the following polynomials and polynomial identities:

Polynomials as Computed by Honest Parties

$$\begin{aligned}
\mathsf{b}(\mathsf{X}) &= \sum_{i=0}^{n-1} bit_i \cdot \mathsf{L_i}(\mathsf{X}) \\
\mathsf{pkx}(\mathsf{X}) &= \sum_{i=0}^{n-2} pkx_i \cdot \mathsf{L_i}(\mathsf{X}) \\
\mathsf{pky}(\mathsf{X}) &= \sum_{i=0}^{n-2} pky_i \cdot \mathsf{L_i}(\mathsf{X}) \\
\mathsf{kaccx}(\mathsf{X}) &= \sum_{i=0}^{n-1} kaccx_i \cdot \mathsf{L_i}(\mathsf{X}) \\
\mathsf{kaccy}(\mathsf{X}) &= \sum_{i=0}^{n-1} kaccy_i \cdot \mathsf{L_i}(\mathsf{X}),
\end{aligned}$$

where $(pkx_0, \ldots, pkx_{n-2})$ and $(pky_0, \ldots, pky_{n-2})$ are computed such that $\forall i \in \{0, \ldots, n-2\}$, pk_i is interpreted as a pair (pkx_i, pky_i) with its components in $\mathbb{F}$; we also have $(kaccx_0, kaccy_0) = (h_x, h_y)$ and $(kaccx_{i+1}, kaccy_{i+1}) = (kaccx_i, kaccy_i) \oplus bit_i(pkx_i, pky_i)$, $\forall i < n-1$. Note that in the last relation bit_i is not interpreted as a field element anymore but as a binary bit.

Polynomial Identities

$$
\begin{aligned}
id_1(X) &= (X - \omega^{n-1}) \cdot [b(X) \cdot ((kaccx(X) - pkx(X))^2 \cdot (kaccx(X) + pkx(X) + kaccx(\omega \cdot X)) - \\
&\quad - (pky(X) - kaccy(X))^2) + (1 - b(X)) \cdot (kaccy(\omega \cdot X) - kaccy(X))] \\
id_2(X) &= (X - \omega^{n-1}) \cdot [b(X) \cdot ((kaccx(X) - pkx(X)) \cdot (kaccy(\omega \cdot X) + kaccy(X)) - \\
&\quad - (pky(X) - kaccy(X)) \cdot (kaccx(\omega \cdot X) - kaccx(X))) + (1 - b(X)) \cdot (kaccx(\omega \cdot X) - kaccx(X))] \\
id_3(X) &= (kaccx(X) - h_x) \cdot L_0(X) + (kaccx(X) - (h \oplus apk)_x) \cdot L_{n-1}(X) \\
id_4(X) &= (kaccy(X) - h_y) \cdot L_0(X) + (kaccy(X) - (h \oplus apk)_y) \cdot L_{n-1}(X) \\
id_5(X) &= b(X)(1 - b(X)).
\end{aligned}
$$

Note that polynomial identity $id_5(X)$ is not needed for defining ranged polynomial protocols for $\mathcal{R}_{\mathsf{ba}}^{incl}$, however it is included here to ease presentation and for proofs consistency for the ranged polynomial protocols in the following two sections.

H-ranged Polynomial Protocol for Conditional Packed Accountable Relation $\mathcal{R}_{\mathsf{ba}}^{incl}$

In the following, we describe H-ranged polynomial protocol $\mathscr{P}_{\mathsf{ba}}$ for conditional relation $\mathcal{R}_{\mathsf{ba}}^{incl}$. Protocol $\mathscr{P}_{\mathsf{ba}}$ describes the interaction of three parties, the prover $\mathcal{P}_{poly}$, the verifier $\mathcal{V}_{poly}$ and the trusted third party $\mathcal{I}$ in accordance to Definition 25 from section A.

Protocol $\mathscr{P}_{\mathsf{ba}}$

$\mathcal{P}_{poly}$ and $\mathcal{V}_{poly}$ know public input $\mathbf{bit} \in \mathbb{B}^n$, $\mathbf{pk} \in (\mathbb{F}^2)^{n-1}$ and $apk \in (\mathbb{F})^2$ which are interpreted as per their respective domains.

1. $\mathcal{V}_{poly}$ computes $b(X)$, $pkx(X)$, $pky(X)$.
2. $\mathcal{P}_{poly}$ sends polynomials $kaccx(X)$ and $kaccy(X)$ to $\mathcal{I}$.
3. $\mathcal{V}_{poly}$ asks $\mathcal{I}$ to check whether the following polynomial relations hold over range H
$$id_i(X) = 0, \forall i \in [4].$$
4. $\mathcal{V}_{poly}$ accepts if $\mathcal{I}$'s checks verify.

We show that protocol $\mathscr{P}_{\mathsf{ba}}$ is an H-ranged polynomial protocol for conditional relation $\mathcal{R}_{\mathsf{ba}}^{incl}$. For this, we first prove that:

Claim 6. *Assume that $\forall i < n-1$ such that $bit_i = 1$, $pk_i = (pkx_i, pky_i) \in \mathbb{G}_{1,inn}$. If polynomial identities $id_i(X) = 0, \forall i \in [5]$, hold over range H and and the polynomial $b(X)$ has been constructed via interpolation on H such that $b(\omega^i) = bit_i, \forall i < n$ then*
$bit_i \in \mathbb{B} = \{0, 1\} \subset \mathbb{F}, \forall i < n$
$(kaccx_0, kaccy_0) = (h_x, h_y)$,
$(kaccx_{n-1}, kaccy_{n-1}) = (h_x, h_y) \oplus (apk_x, apk_y)$,
$(kaccx_{i+1}, kaccy_{i+1}) = (kaccx_i, kaccy_i) \oplus bit_i(pkx_i, pky_i)$, $\forall i < n-1$, where in the last relation bit_i should not be interpreted as a field element but as a binary bit.

Proof. Everything but the last property in the claim is easy to derive from polynomial identities $id_3(X) = 0, id_4(X) = 0, id_5(X) = 0$ holding over H.

In order to prove the remaining property, we remind the incomplete addition formulae for curve points in affine coordinates, over elliptic curve in short Weierstrasse form and state:

Observation: Suppose that $bit \in \{0,1\}$, (x_1, y_1) is a point on an elliptic curve in short Weierstrasse form, and, if $bit = 1$, so is (x_2, y_2). We claim that the following equations:

$$bit((x_1 - x_2)^2(x_1 + x_2 + x_3) - (y_2 - y_1)^2) + (1 - bit)(y_3 - y_1) = 0 \ (*)$$
$$bit((x_1 - x_2)(y_3 + y_1) - (y_2 - y_1)(x_3 - x_1)) + (1 - bit)(x_3 - x_1) = 0 \ (**)$$

hold if and only if one of the following three conditions hold

1. $bit = 1$ and $(x_1, y_1) \oplus (x_2, y_2) = (x_3, y_3)$ and $x_1 \neq x_2$
2. $bit = 0$ and $(x_3, y_3) = (x_1, y_1)$
3. $bit = 1$ and $(x_1, y_1) = (x_2, y_2)$[4].

It is easy to see that each of the conditions 1,2,3 above implies equations $(*)$ and $(**)$. For the implication in the opposite direction, if we assume that $(*)$ and $(**)$ hold, then

Case a: For $bit = 0$, the first term of each equation $(*)$ and $(**)$ vanishes, leaving us with $y_3 - y_1 = 0$ and $x_3 - x_1 = 0$ which are equivalent to condition 2.

Case b: For $bit = 1$ and $x_1 = x_2$, by simple substitution in $(*)$ and $(**)$, we obtain $y_1 = y_2$, i.e., condition 3.

Case c: For $bit = 1$ and $x_1 \neq x_2$, then we can substitute

$$\beta = \frac{y_2 - y_1}{x_2 - x_1}$$

into equations $(*)$ and $(**)$, leaving us with

$$x_1 + x_2 + x_3 = \beta^2 \text{ and } y_3 + y_1 = \beta(x_3 - x_1).$$

which are the usual formulae for short Weierstrass form addition of affine coordinate points when $x_1 \neq x_2$ so this is equivalent to condition 1.

We apply the above *Observation* by noticing that if $id_1(X)$ and $id_2(X)$ hold over H, then $(*)$ and $(**)$ hold with (x_1, y_1) substituted by $(kaccx_i, kaccy_i)$, (x_2, y_2) substituted by (pkx_i, pky_i), (x_3, y_3) substituted by $(kaccx_{i+1}, kaccy_{i+1})$ and bit substituted by bit_i for $0 \leq i \leq n-2$, where bit_i should not be interpreted as a field element but as binary bit. Moreover, since $(kaccx_0, kaccy_0) = (h_x, h_y) \in E_{inn} \setminus \mathbb{G}_{1,inn}$ and if $(pkx_i, pky_i) \in \mathbb{G}_{1,inn}$ whenever $bit_i = 1$, then $\forall i < n-1$ equations $(*)$ and $(**)$ obtained after the substitution defined above are equivalent to either condition 1 or condition 2, but never condition 3, so the result of the sum (i.e., $(kaccx_{i+1}, kaccy_{i+1})$, $0 \leq i \leq n-2$) is, by induction, at each step a well-defined point on the curve and this concludes our proof. □

Corollary 7. *Assume $\forall i < n-1$ such that $bit_i = 1$, $pk_i = (pkx_i, pky_i) \in \mathbb{G}_{1,inn}$. If the polynomial identities $id_i(X) = 0, \forall i \in [4]$, hold over range H and $bit_i \in \mathbb{B}$, $\forall i < n-1$ and $b(X) = \sum_{i=0}^{n-1} bit_i \cdot L_i(X)$ then:*
$(kaccx_0, kaccy_0) = (h_x, h_y)$,
$(kaccx_{n-1}, kaccy_{n-1}) = (h_x, h_y) \oplus (apk_x, apk_y)$,
$(kaccx_{i+1}, kaccy_{i+1}) = (kaccx_i, kaccy_i) \oplus bit_i(pkx_i, pky_i)$, $\forall i < n-1$, where in the last relation bit_i should not be interpreted as a field element but as a binary bit.

[4]Note that under condition 3, (x_3, y_3) can be any point whatsoever, maybe not even on the curve. The same holds true for (x_2, y_2) under the condition 2.

Proof. The proof follows trivially from the more general result stated by Claim 6. □

Lemma 8. *$\mathscr{P}_{\mathsf{ba}}$ as described above is an H-ranged polynomial protocol for conditional relation $\mathcal{R}_{\mathsf{ba}}^{incl}$.*

Proof. It is easy to see that perfect completeness holds. Indeed, if $(\mathbf{bit}, \mathbf{pk}, apk) \in \mathcal{R}_{\mathsf{ba}}^{incl}$ holds, meaning that $\mathbf{bit} \in \mathbb{B}^n$ and $\mathbf{pk} \in \mathbb{G}_{1,inn}^{n-1}$ and $apk = \sum_{i=0}^{n-2}[bit_i] \cdot pk_i$ hold, then it is easy to see that the honest prover $\mathcal{P}_{poly}$ in $\mathscr{P}_{\mathsf{ba}}$ will convince the honest verifier $\mathcal{V}_{poly}$ in $\mathscr{P}_{\mathsf{ba}}$ to accept with probability 1.
Regarding knowledge-soundness, if the verifier $\mathcal{V}_{poly}$ in $\mathscr{P}_{\mathsf{ba}}$ accepts, then the extractor $\mathcal{E}$ does not have to do anything as the relation $\mathcal{R}_{\mathsf{ba}}^{incl}$ does not have a witness. However, we have to prove that if $\mathbf{pk} \in \mathbb{G}_{1,inn}^{n-1}$ and the verifier in $\mathscr{P}_{\mathsf{ba}}$ accepts, then $(\mathbf{bit}, \mathbf{pk}, apk) \in \mathcal{R}_{\mathsf{ba}}^{incl}$ holds, which given our definition for conditional relation is equivalent to proving that $apk = \sum_{i=0}^{n-2}[bit_i] \cdot pk_i$ holds. This is indeed the case due to Corollary 7.

□

3.2 Packed Accountable Ranged Polynomial Protocol

In the following, we denote by $\mathbb{F}_{|\mathsf{block}|}$ the subset of field elements in $\mathbb{F}$ that can be represented using at most block bits, i.e., the set $\{0, \ldots, 2^{\mathsf{block}-1}\}$, where block has been defined in section 2.7.
Our conditional packed accountable relation $\mathcal{R}_{\mathsf{pa}}^{incl}$ and the corresponding H-ranged polynomial protocol $\mathscr{P}_{\mathsf{pa}}$ are defined as follows:

Conditional Packed Accountable Relation $\mathcal{R}_{\mathsf{pa}}^{incl}$

$$\mathcal{R}_{\mathsf{pa}}^{incl} = \{(\mathbf{pk} \in (\mathbb{F}^2)^{n-1}, \mathbf{b}' \in \mathbb{F}_{|\mathsf{block}|}^{\frac{n}{\mathsf{block}}}, apk \in \mathbb{F}^2; \mathbf{bit}) : apk = \sum_{i=0}^{n-2}[bit_i] \cdot pk_i \mid \mathbf{pk} \in \mathbb{G}_{1,inn}^{n-1} \wedge$$

$$\wedge\, \mathbf{bit} \in \mathbb{B}^n \wedge b'_j = \sum_{i=0}^{\mathsf{block}-1} 2^i \cdot bit_{\mathsf{block} \cdot j + i}, \forall j < \frac{n}{\mathsf{block}}\}$$

where $\mathbf{b}' = (b'_0, \ldots, b'_{\frac{n}{\mathsf{block}}-1})$.
We define new polynomials and polynomial identities:

New Polynomials as Computed by Honest Parties

$$aux(X) = \sum_{i=0}^{n-1} aux_i \cdot L_i(X)$$

$$c_a(X) = \sum_{i=0}^{n-1} c_{a,i} \cdot L_i(X)$$

$$acc_a(X) = \sum_{i=0}^{n-1} acc_{a,i} \cdot L_i(X)$$

where $aux_i = 1 \in \mathbb{F}$ if i is divisible with block and $aux_i = 0 \in \mathbb{F}$ otherwise, $\forall i < n$
and $c_{a,i} = 2^k \cdot r^j$, $k = i \mod \mathsf{block}$, $j = i \div \mathsf{block}$, $\forall i < n$ ($r \in \mathbb{F}$ is introduced in protocol $\mathscr{P}_{\mathsf{pa}}$) and $acc_{a,i}$ are components of the vector $(0, bit_0 \cdot c_{a,0}, bit_0 \cdot c_{a,0} + bit_1 \cdot c_{a,1}, \ldots, \sum_{i=0}^{n-2} bit_i \cdot c_{a,i})$, where $bit_0, \ldots, bit_{n-1}$ represent the first n bits (however, we interpret them as elements in $\mathbb{B}$) of the concatenation of the binary representation of $b'_0, \ldots, b'_{\frac{n}{\mathsf{block}}-1}$.[5] Note that with this definition of vector $(bit_0, \ldots, bit_{n-1})$, the definition of $b(X)$ remains the same as in section 3.1.

[5] As part of a correct public input for relation $\mathcal{R}_{\mathsf{pa}}^{incl}$, each field element in the set $\{b'_0, \ldots, b'_{\frac{n}{\mathsf{block}}-1}\}$ is at most block binary bits long. If any such field element has fewer than block bits, then the honest prover will pad it with 0s starting from the most significant bit up to a total individual length of block bits.

New Polynomial Identities

$$id_6(X) = c_a(\omega \cdot X) - c_a(X) \cdot (2 + (\frac{r}{2^{\mathsf{block}-1}} - 2) \cdot aux(\omega \cdot X)) - (1 - r^{\frac{n}{\mathsf{block}}}) \cdot L_{n-1}(X).$$
$$id_7(X) = acc_a(\omega \cdot X) - acc_a(X) - b(X) \cdot c_a(X) + \mathsf{sum} \cdot L_{n-1}(X),$$

where sum is a field element known to both $\mathcal{P}_{poly}$ and $\mathcal{V}_{poly}$ and will be defined below.

H-ranged Polynomial Protocol for Conditional Packed Accountable Relation $\mathcal{R}_{\mathsf{pa}}^{incl}$

In the following, we describe H-ranged polynomial protocol $\mathscr{P}_{\mathsf{pa}}$ for conditional relation $\mathcal{R}_{\mathsf{pa}}^{incl}$.

Protocol $\mathscr{P}_{\mathsf{pa}}$

$\mathcal{P}_{poly}$ and $\mathcal{V}_{poly}$ know public inputs $\mathbf{b'} \in \mathbb{F}^{\frac{n}{\mathsf{block}}}_{|\mathsf{block}|}$ and $\mathbf{pk} \in (\mathbb{F}^2)^{n-1}$ and $apk \in \mathbf{F}^2$ which are interpreted as per their respective domains.

1. $\mathcal{V}_{poly}$ computes $pkx(X)$, $pky(X)$ and $aux(X)$.
2. $\mathcal{P}_{poly}$ sends polynomials $b(X)$, $kaccx(X)$ and $kaccy(X)$ to $\mathcal{I}$.
3. $\mathcal{V}_{poly}$ replies with a random value r chosen from $\mathbb{F}$.
4. $\mathcal{V}_{poly}$ computes sum as $\sum_{j=0}^{\frac{n}{\mathsf{block}}-1} b'_j \cdot r^j$.[6]
5. $\mathcal{P}_{poly}$ sends polynomials $c_a(X)$ and $acc_a(X)$ to $\mathcal{I}$.
6. $\mathcal{V}_{poly}$ asks $\mathcal{I}$ to check whether the following polynomial relations hold over range H:
$$id_i(X) = 0, \forall i \in [7].$$
7. $\mathcal{V}_{poly}$ accepts if $\mathcal{I}$'s checks verify.

We show that protocol $\mathscr{P}_{\mathsf{pa}}$ is an H-ranged polynomial protocol for conditional relation $\mathcal{R}_{\mathsf{pa}}^{incl}$. First, we prove the following:

Claim 9. *If the polynomial identities $id_6(X) = 0, id_7(X) = 0$ hold over range H, then, e.w.n.p. , we have $c_{a,i} = 2^{i \mod \mathsf{block}} \cdot r^{i \div \mathsf{block}}$, $\forall i < n$ and $\mathsf{sum} = \sum_{i=0}^{n-1} b_i \cdot c_{a,i}$, where $b_i = b(\omega^i), \forall i < n$. If, additionally, identity $id_5(X) = 0$ holds over H, r has been randomly chosen in $\mathbb{F}$, $\mathsf{sum} = \sum_{j=0}^{\frac{n}{\mathsf{block}}-1} b'_j r^j$ (as computed by $\mathcal{V}_{poly}$) and $bit_i \in \mathbb{B}, \forall i < n$ and $b'_j = \sum_{k=0}^{\mathsf{block}-1} 2^k \cdot bit_{\mathsf{block} \cdot j + k}, \forall 0 \leq j \leq \frac{n}{\mathsf{block}} - 1$ (due to the input $(b'_0, \ldots, b'^{\frac{n}{\mathsf{block}}-1})$ being interpreted by the verifier $\mathcal{V}_{poly}$ as in $\mathbb{F}^{\frac{n}{\mathsf{block}}}_{|\mathsf{block}|}$), then e.w.n.p. , $b_i = bit_i, \forall i < n$.*

Proof. To prove the first part of the claim, assume by contradiction that $c_{a,0} = k \neq 1$. Then, by induction, since $id_6(X) = 0$ on H,

$$c_{a,i} = k \cdot 2^{i \mod \mathsf{block}} \cdot r^{i \div \mathsf{block}}, \forall 0 < i < n.$$

Additionally, the property

$$c_{a,0} = c_{a,n-1} \cdot (2 + (\frac{r}{2^{\mathsf{block}-1}} - 2) \cdot 1) + (1 - r^{\frac{n}{\mathsf{block}}}) \tag{1}$$

holds (again, from $id_6(X) = 0$ on H). However, substituting $c_{a,0} = k$ and $c_{a,n-1} = k \cdot 2^{\mathsf{block}-1} \cdot r^{\frac{n}{\mathsf{block}}-1}$ in (1), we obtain $k = k \cdot 2^{\mathsf{block}-1} \cdot r^{\frac{n}{\mathsf{block}}-1} \cdot \frac{r}{2^{\mathsf{block}-1}} + 1 - r^{\frac{n}{\mathsf{block}}}$ which is equivalent to $k(1 - r^{\frac{n}{\mathsf{block}}}) = 1 - r^{\frac{n}{\mathsf{block}}}$,

[6] Note that if $b'_j = \sum_{k=0}^{\mathsf{block}-1} 2^k \cdot bit_{\mathsf{block} \cdot j + k}$, $\forall j < \frac{n}{\mathsf{block}}$ and $bit_i \in \mathbb{B}, \forall i < n$, then $\sum_{i=0}^{n-1} 2^{i \mod \mathsf{block}} \cdot r^{i \div \mathsf{block}} \cdot bit_i = \sum_{j=0}^{\frac{n}{\mathsf{block}}-1} (\sum_{i=0}^{\mathsf{block}-1} 2^k \cdot bit_{\mathsf{block} \cdot j + k}) \cdot r^j = \sum_{j=0}^{\frac{n}{\mathsf{block}}-1} b'_j \cdot r^j$.

and, due to Schwartz-Zippel Lemma and the fact that degree n is negligibly smaller compared to the size of $\mathbb{F}$, this implies e.w.n.p. $k = 1$ thus contradiction, so the values $c_{a,i}$ have indeed the claimed form.
Next, by expanding $id_7(X) = 0$ over H, the following holds

$$\begin{aligned} acc_{a,1} &= acc_{a,0} + b_0 \cdot c_{a,0} \\ acc_{a,2} &= acc_{a,1} + b_1 \cdot c_{a,1} \\ &\dots \\ acc_{a,n-1} &= acc_{a,n-2} + b_{n-2} \cdot c_{a,n-2} \\ acc_{a,0} &= acc_{a,n-1} + b_{n-1} \cdot c_{a,n-1} - \mathsf{sum}. \end{aligned}$$

By summing together the LHS and, respectively, the RHS of the equalities above and reducing the equal terms, we obtain $\mathsf{sum} = \sum_{i=0}^{n-1} b_i \cdot c_{a,i}$.

For the second part of the claim, since $id_5(X) = 0$ holds over H then $b_i = b(\omega^i) \in \mathbb{B}, \forall i \leq n-1$. Finally, from verifier's computation and from the first part of the claim we have

$$\sum_{j=0}^{\frac{n}{\mathsf{block}}-1} b'_j r^j = \mathsf{sum} = \sum_{i=0}^{n-1} b_i \cdot c_{a,i} = \sum_{i=0}^{n-1} b_i \cdot 2^{i \mod \mathsf{block}} \cdot r^{i \div \mathsf{block}} =$$

$$= \sum_{j=0}^{\frac{n}{\mathsf{block}}-1} \Big(\sum_{k=0}^{\mathsf{block}-1} 2^k \cdot b_{\mathsf{block} \cdot j + k} \Big) \cdot r^j = \sum_{j=0}^{\frac{n}{\mathsf{block}}-1} b''_j r^j, \tag{2}$$

where $\forall j, b''_j$ are field elements equal to the binary representation that uses contiguous blocks of block components from the bitmask $(b_0, \dots, b_{n-1})$. Since both the LHS and the RHS of relation (2) represent two ways of computing sum as an inner product of a vector of field elements (on one hand, $(b'_0, \dots, b'_{\frac{n}{\mathsf{block}}-1})$, on the other hand, $(b''_0, \dots, b''_{\frac{n}{\mathsf{block}}-1})$) with the vector $(1, r, \dots, r^{\frac{n}{\mathsf{block}}-1})$, where r has been chosen at random, by the small exponents test [40], we obtain that e.w.n.p. $b''_j = b'_j, \forall 0 \leq j \leq \frac{n}{\mathsf{block}} - 1$. Finally, if we equate the bit representation in $\mathbb{F}$ (i.e., using field elements from $\mathbb{B}$) of field elements b''_j and $b'_j, \forall 0 \leq j \leq \frac{n}{\mathsf{block}} - 1$ and remember that, by verifier's check or by construction, respectively, each such field element has no more that block binary bits, we can conclude that e.w.n.p. $b_i = bit_i, \forall i < n$. □

Lemma 10. *$\mathscr{P}_{\mathsf{pa}}$ as described above is an H-ranged polynomial protocol for conditional relation $\mathcal{R}^{incl}_{\mathsf{pa}}$.*

Proof. It is easy to see that perfect completeness holds. Indeed, if $(\mathbf{b'}, \mathbf{pk}, apk, \mathbf{bit}) \in \mathcal{R}^{incl}_{\mathsf{pa}}$, meaning that $\mathbf{pk} \in \mathbb{G}^{n-1}_{1,inn}$ and $\mathbf{bit} \in \mathbb{B}^n$ and $apk = \sum_{i=0}^{n-2} [bit_i] \cdot pk_i$ and $b'_j = \sum_{i=0}^{\mathsf{block}-1} 2^i \cdot bit_{\mathsf{block} \cdot j + i}, \forall j < \frac{n}{\mathsf{block}}$ hold then it is easy to see that the honest prover $\mathcal{P}_{poly}$ in $\mathscr{P}_{\mathsf{pa}}$ will convince the honest verifier $\mathcal{V}_{poly}$ in $\mathscr{P}_{\mathsf{pa}}$ to accept with probability 1.
Regarding knowledge-soundness, if the verifier $\mathcal{V}_{poly}$ in $\mathscr{P}_{\mathsf{pa}}$ accepts, then the extractor $\mathcal{E}$ sets $(bit_0, \dots, bit_{n-1})$ as the vector of evaluations over H of polynomial $b(X)$ sent by $\mathcal{P}_{poly}$ to $\mathcal{I}$. Next, we prove that if $(pk_0, \dots, pk_{n-2}) \in \mathbb{G}^{n-1}_{1,inn}$ and the verifier in $\mathscr{P}_{\mathsf{pa}}$ accepts, then

$$((b'_0, \dots, b'_{\frac{n}{\mathsf{block}}-1}), (pk_0, \dots, pk_{n-2}), apk, (bit_0, \dots, bit_{n-1})) \in \mathcal{R}^{incl}_{\mathsf{pa}},$$

which is equivalent to proving that $apk = \sum_{i=0}^{n-2} [bit_i] \cdot pk_i$ and $\mathbf{bit} \in \mathbb{B}^n$ and

$$b'_j = \sum_{i=0}^{\mathsf{block}-1} 2^i \cdot bit_{\mathsf{block} \cdot j + i}, \forall j < \frac{n}{\mathsf{block}}.$$

According to Claim 9 and Corollary 7 this indeed holds e.w.n.p. □

3.3 Counting Ranged Polynomial Protocol

In the following relation, apk is the aggregated public key of at least s and at most $s+1$ public keys. Hence we interpret s as a threshold on the number of public keys included in the aggregated public key. Since bit_{n-1} as the last component of the bitmask witness does not correspond to any public key and we have to account for the fact that bit_{n-1} may be $1 \in \mathbb{F}$, relation $\mathcal{R}_{\mathsf{c}}^{incl}$ includes the off-by-one constraint $\sum_{i=0}^{n-1} bit_i = s+1$.

Conditional Counting Relation $\mathcal{R}_{\mathsf{c}}^{incl}$

$$\mathcal{R}_{\mathsf{c}}^{incl} = \{(\mathbf{pk} \in (\mathbb{F}^2)^{n-1}, s \in \mathbb{F}^2, apk \in \mathbb{F}^2; \mathbf{bit}) : apk = \sum_{i=0}^{n-2} [bit_i] \cdot pk_i \mid \mathbf{pk} \in \mathbb{G}_{1,inn}^{n-1} \wedge$$
$$\wedge \ \mathbf{bit} \in \mathbb{B}^n \ \wedge \ \sum_{i=0}^{n-1} bit_i = s+1\}$$

The new polynomials and polynomial identities required in this section are:

New Polynomial as Computed by Honest Parties

$$acc_{vt}(X) = \sum_{i=0}^{n-1} acc_{vt,i} \cdot L_i(X),$$

where $acc_{vt,i}$ are the components of the vector $(0, bit_0, bit_0 + bit_1, \ldots, \sum_{i=0}^{n-2} bit_i)$, $\forall i < n$.

New Polynomial Identities

$$id_8(X) = acc_{vt}(\omega \cdot X) - acc_{vt}(X) - b(X) + (s+1) \cdot L_{n-1}(X),$$

H-ranged Polynomial Protocol for Conditional Counting Relation $\mathcal{R}_{\mathsf{c}}^{incl}$

Protocol $\mathscr{P}_{\mathsf{c}}$

$\mathcal{P}_{poly}$ and $\mathcal{V}_{poly}$ know public input $s \in \mathbb{F}^2$, $\mathbf{pk} \in (\mathbb{F}^2)^{n-1}$ and $apk \in \mathbb{F}^2$ which are interpreted as per their respective domains.

1. $\mathcal{V}_{poly}$ computes $pkx(X)$, $pky(X)$.
2. $\mathcal{P}_{poly}$ sends polynomials $b(X)$, $kaccx(X)$, $kaccy(X)$, $acc_{vt}(X)$ to $\mathcal{I}$.
3. $\mathcal{V}_{poly}$ asks $\mathcal{I}$ to check whether the following polynomial relations hold over range H:
$$id_i(X) = 0, \forall i \in [5] \ and \ id_8(X) = 0.$$
4. $\mathcal{V}_{poly}$ accepts if all of $\mathcal{I}$'s checks verify.

We show that protocol $\mathscr{P}_{\mathsf{c}}$ is an H-ranged polynomial protocol for conditional relation $\mathcal{R}_{\mathsf{c}}^{incl}$.

Lemma 11. *$\mathscr{P}_{\mathsf{c}}$ as described above is an H-ranged polynomial protocol for conditional relation $\mathcal{R}_{\mathsf{c}}^{incl}$.*

Proof. It is easy to see that perfect completeness holds. Indeed, if $(\mathbf{bit}, \mathbf{pk}, apk) \in \mathcal{R}_{\mathsf{c}}^{incl}$ holds, meaning that $\mathbf{bit} \in \mathbb{B}^n$ and $\mathbf{pk} \in \mathbb{G}_{1,inn}^{n-1}$ and $apk = \sum_{i=0}^{n-2} [bit_i] \cdot pk_i$ and $\sum_{i=0}^{n-1} bit_i = s+1$ hold, then it is easy to see that the honest prover $\mathcal{P}_{poly}$ in $\mathscr{P}_{\mathsf{c}}$ will convince the honest verifier $\mathcal{V}_{poly}$ in $\mathscr{P}_{\mathsf{c}}$ to accept with probability 1.

Regarding knowledge-soundness, if the verifier $\mathcal{V}_{poly}$ in $\mathscr{P}_{\mathsf{c}}$ accepts, then we construct the extractor $\mathcal{E}$ in the following way. Using the polynomial $b(X)$ which was part of the messages from $\mathcal{P}_{poly}$ to $\mathcal{I}$ and evaluating it at the elements of the set H, $\mathcal{E}$ obtains evaluation vector $\mathbf{bit} = (b(1), \ldots, b(\omega^{n-1}))$ which, in the following, we denote as $(bit_0, \ldots, bit_{n-1}) \in \mathbb{F}^n$.
Next, we show that if $(pk_0, \ldots, pk_{n-2}) \in \mathbb{G}_{1,inn}^{n-1}$ holds and the verifier in $\mathscr{P}_{\mathsf{c}}$ accepts, then

$$((pk_0, \ldots, pk_{n-2}), s, apk, (bit_0, \ldots, bit_{n-1})) \in \mathcal{R}_{\mathsf{c}}^{incl},$$

which is equivalent to proving that $apk = \sum_{i=0}^{n-2}[bit_i] \cdot pk_i$ and $\mathbf{bit} \in \mathbb{B}^n$ and $\sum_{i=0}^{n-1} bit_i = s + 1$. First, since $id_8(X) = 0$ holds over H, we can expand that as follows:

$$\begin{aligned} acc_{vt,1} &= acc_{vt,0} + bit_0 \\ acc_{vt,2} &= acc_{vt,1} + bit_1 \\ &\ldots \\ acc_{vt,n-1} &= acc_{vt,n-2} + bit_{n-2} \\ acc_{vt,0} &= acc_{vt,n-1} + bit_{n-1} - (s+1). \end{aligned}$$

By summing together the LHS and, respectively, the RHS of the equalities above and reducing the equal terms, we obtain $s + 1 = \sum_{i=0}^{n-1} bit_i$.
Second, since it holds over H that $id_i(X) = 0$, $\forall i \in [5]$ and $b(\omega^i) = bit_i, \forall i < n$ (by the definition of $\mathcal{E}$), the properties concluded in Claim 6 hold. Combining the two proof steps above, we obtain the desired conclusion. □

3.4 Two-Steps PLONK-Based Compiler for Hybrid Model SNARKs with Mixed Inputs

In the following we present a two-steps PLONK-based compilation technique from ranged polynomial protocols for conditional NP relations (formal definition in appendix A) to hybrid model SNARKs as per definition 5 such that the conditional NP relations that define the SNARKs we compile in the second step contain both (polynomial) commitments and vectors of field elements as public inputs. For completeness and as an example, in appendix B we give the full rolled-out hybrid model SNARK protocol $\mathscr{P}_{\mathsf{pa}}^{h}$ for relation $\mathcal{R}_{\mathsf{pa},com}^{incl}$, where we define $\mathscr{P}_{\mathsf{pa}}^{h}$ and $\mathcal{R}_{\mathsf{pa},com}^{incl}$ in Step 2 of our compiler below. By using just the first step of our compiler which is equivalent (modulo some more clarifications necessary for our use cases) to the original PLONK compiler, one would not be able to obtain SNARKs with mixed public inputs consisting of both vectors of field elements and also (polynomial) commitments. In turn, this type of NP relations with mixed inputs is crucial for designing and proving the security of our accountable light clients in section 4.

Step 1 (PLONK compiler - from polynomial protocols to SNARKs):

Our first step applies the PLONK compiler [11]. More precisely, we compile the information theoretical ranged polynomial protocols $\mathscr{P}_{\mathsf{ba}}$, $\mathscr{P}_{\mathsf{pa}}$ and $\mathscr{P}_{\mathsf{c}}$ for relations $\mathcal{R}_{\mathsf{ba}}^{incl}$, $\mathcal{R}_{\mathsf{pa}}^{incl}$ and $\mathcal{R}_{\mathsf{c}}^{incl}$ respectively (as defined in sections 3.1, 3.2, 3.3) into computationally secure protocols against AGM adversaries. The resulting protocols are, in fact, SNARKs. In order to keep in sync with PLONK notation, we denote the resulting SNARK protocols by $\mathscr{P}_{\mathsf{ba}}^{*}$, $\mathscr{P}_{\mathsf{pa}}^{*}$ and $\mathscr{P}_{\mathsf{c}}^{*}$, respectively. In fact, we can define this compilation step in a general way, for any ranged polynomial protocols for relations (as per definition in appendix A). In order to do that we need:

- The batched version of KZG polynomial commitments [19] described in section 3 of PLONK [11].[7]

[7] In fact, one can replace the use of KZG polynomial commitments with any binding polynomial commitment that has knowledge-soundness, including non-homomorphic polynomial commitments, such as FRI-based polynomial commitments (e.g., RedShift [41]). If the optimisation gained from PLONK linearisation technique is a goal, then, with minimal changes one can use any homomorphic polynomial commitment, e.g., the discrete logarithm based polynomial commitment from Halo [42].

- A general compilation technique: such a technique has been already defined in lemma 4.7 of PLONK; combined with lemma 4.5 from PLONK this technique can be applied with minor adaptations (this includes the corresponding technical measures) to the notion of ranged polynomial protocols as defined in appendix A.
- So far, both the ranged polynomial protocols for relations and the protocols resulted after the first compilation step have been explicitly defined as interactive protocols. In order to obtain the non-interactive version of the latter (essentially the N in SNARK) one has to apply the Fiat-Shamir transform [43], [44], [45].

Let $\mathcal{R}$ be a (conditional) NP relation, let $\mathscr{P}_{\mathcal{R}}$ be a ranged polynomial protocol for relation $\mathcal{R}$ and let $\mathscr{P}^*_{\mathcal{R}}$ be the SNARK compiled from $\mathscr{P}_{\mathcal{R}}$ using the PLONK compiler (as summarised above). Going into more detail, the above compilation technique requires the SNARK prover of $\mathscr{P}^*_{\mathcal{R}}$ to compute polynomial commitments to all polynomials that the prover $\mathcal{P}_{poly}$ in $\mathscr{P}_{\mathcal{R}}$ sent to the ideal party $\mathcal{I}$. Analogously, it requires the SNARK verifier of $\mathscr{P}^*_{\mathcal{R}}$ to compute polynomial commitments to all pre-processed polynomials[8] as well polynomial commitments to polynomials the verifier $\mathcal{V}_{poly}$ in $\mathscr{P}_{\mathcal{R}}$ sent to the ideal party $\mathcal{I}$. Then, the SNARK prover sends the SNARK verifier openings to all the polynomial commitments computed by him as well as the polynomial commitments computed by the SNARK verifier. The SNARK prover additionally sends the corresponding batched proofs for polynomial commitment openings. In turn, the SNARK verifier accepts or rejects based on the result of the verification of the batched polynomial commitment scheme.

A more efficient compilation technique exists which reduces the number of polynomial commitments and alleged polynomial commitments openings (i.e., both group elements and field elements) sent by the SNARK prover to the SNARK verifier; this, in turn, reduces the size of the SNARK proof. This technique is called linearisation and is described, at a high level, after Lemma 4.7 in PLONK. The existing description however covers only the SNARK prover side and it does not detail the SNARK verifier side so in the following we cover that.

By functionality, the vectors that are handled by the the verifier $\mathcal{V}_{poly}$ are of two types: pre-processed vectors and public input vectors. These two types of vectors are used by $\mathcal{V}_{poly}$ to obtain, via interpolation over the range on which the respective range polynomial protocol is defined, pre-processed polynomials (as used in the definition 25, e.g., polynomial $aux(X)$ used in section 3.2) and public-inputs-derived polynomials (e.g., polynomials $pkx(X)$ and $pky(X)$ used in sections 3.1, 3.2, 3.3 and polynomial $b(X)$ used in section 3.1). The efficient linearisation technique allows the SNARK verifier to reduce the number of polynomial commitments it has to compute compared to the general PLONK compiler in the following way. Instead of having to compute polynomial commitments to all polynomials $\mathcal{V}_{poly}$ sends to $\mathcal{I}$ (including any corresponding pre-processed polynomials), the SNARK verifier computes polynomial evaluations at one or multiple random points (as per the linearisation step specific requirements) for all the polynomials that are either easy to evaluate (e.g., polynomial $aux(X)$ used in section 3.2) or all the polynomials that are obtained from vectors that do not take up a large amount of memory (e.g., polynomial $b(X)$ used in section 3.1). For the rest of the polynomials (e.g., $pkx(X)$ and $pky(X)$), the SNARK verifier computes polynomial commitments as before.

We note we can apply all the techniques mentioned above, including the combined prover-and-verifier-side linearisation to compile our three ranged polynomial protocols $\mathscr{P}_{\mathsf{ba}}$, $\mathscr{P}_{\mathsf{pa}}$ and $\mathscr{P}_{\mathsf{c}}$ into the corresponding SNARKs $\mathscr{P}^*_{\mathsf{ba}}$, $\mathscr{P}^*_{\mathsf{pa}}$ and $\mathscr{P}^*_{\mathsf{c}}$, respectively. To conclude this step, we formally state in appendix A, lemma 26 under which condition and how efficiently one can compile ranged polynomial protocols for conditional NP relations (where the public inputs are interpreted as vector of field elements) into hybrid model SNARKs by using only the original PLONK compiler.

Step 2 (Mixed Vector and Commitments as Input for NP Relations and Associated SNARKs):

[8] This is a one-time computation that is reused by the SNARK verifier for all SNARK proofs over the same circuit.

The type of NP relations we have worked with so far as well as the more general PLONK NP relation ([11], section 8.2), do not have the result of cryptographic operations as part of their public input but rather the public inputs are interpreted by honest parties as vectors of field elements. In the following, we show that the SNARKs we have compiled using Step 1 can become, under certain trust assumption, SNARKs for an additional type of NP relation that specifically contains polynomial commitments as part of the input. As detailed in section 4, interpreting our already compiled SNARKs as SNARKs for this additional type of NP relation is essential for modelling and achieving the security properties for our accountable light client systems.

In order to define Step 2 of our compiler, we need first to introduce some notation. To start with, assume a conditional NP relation $\mathcal{R}^c_{vec}$ (according to the notion introduced in section 2.4) of the form:

$$\begin{aligned}\mathcal{R}^c_{vec} = \{(\mathbf{input_1} \in \mathcal{D}_1, \mathbf{input_2} \in \mathcal{D}_2; \mathbf{witness_1}) : \\ p_1(\mathbf{input_1}, \mathbf{input_2}, \mathbf{witness_1}) = 1 \mid c(\mathbf{input_1}) = 1 \wedge \\ \wedge\ p_2(\mathbf{input_1}, \mathbf{input_2}, \mathbf{witness_1}) = 1\},\end{aligned}$$

where $\mathbf{input_1}$ is a set of public input vectors that should be parsed (but not checked) by the honest parties as belonging to some domain $\mathcal{D}_1$. Analogously, for the set of public input vectors $\mathbf{input_2}$ and their respective domain $\mathcal{D}_2$. Finally, $\mathbf{witness_1}$ is a set of witness vectors and c and p_1 and p_2 are predicates. Let $\mathscr{P}_{vec}$ be a ranged polynomial protocol for relation $\mathcal{R}^c_{vec}$. Note that since the condition predicate c applies only to a part of the public input for relation $\mathcal{R}^c_{vec}$ (i.e., $\mathbf{input_1}$), we can apply lemma 26 and Step 1 of our compiler to polynomial protocol $\mathscr{P}_{vec}$.

Next, we make the following assumptions which we call hybrid model assumptions:

- (HMA.1.) The verifier $\mathcal{V}_{poly}$ in $\mathscr{P}_{vec}$ computes polynomials $Q_{1,\mathbf{input_1}}(X), \ldots, Q_{m,\mathbf{input_1}}(X)$ which depend deterministically on $\mathbf{input_1}$ and sends them to $\mathcal{I}$.
- (HMA.2.) The verifier $\mathcal{V}_{poly}$ in $\mathscr{P}_{vec}$ does not use $\mathbf{input_1}$ in any further computation of any other polynomials or values its sends to $\mathcal{I}$.
- (HMA.3.) By evaluating $Q_{1,\mathbf{input_1}}(X), \ldots, Q_{m,\mathbf{input_1}}(X)$ over the range on which the ranged polynomial protocol $\mathscr{P}_{vec}$ is defined one obtains (using some efficiently computable and deterministic transformations) the set of vectors $\mathbf{input_1}$.

Let us denote by $\mathscr{P}^*_{vec}$ the hybrid model SNARK obtained after compiling $\mathscr{P}_{vec}$ using Step 1 of our compiler. Due to assumption (HMA.1.) and according to Step 1 of our compiler, the SNARK verifier in $\mathscr{P}^*_{vec}$ computes

$$Com_1 = Com(Q_{1,\mathbf{input_1}}), \ldots, Com_m = Com(Q_{m,\mathbf{input_1}})$$

which are KZG polynomial commitments to $Q_{1,\mathbf{input_1}}(X), \ldots, Q_{m,\mathbf{input_1}}(X)$. For brevity, we denote the vector $(Com_1, \ldots, Com_m)$ by $\mathbf{Com(input_1)}$ and we denote by $\mathcal{C}$ the set of all KZG polynomial commitments or vectors of such polynomial commitments.

Let us also define the relation:

$$\begin{aligned}\mathcal{R}^c_{vec,com} = \{\mathbf{C} \in \mathcal{C}, \mathbf{input_2} \in \mathcal{D}_2; \mathbf{witness_1}, \mathbf{witness_2}) : \\ p_1(\mathbf{witness_2}, \mathbf{input_2}, \mathbf{witness_1}) = 1 \mid c(\mathbf{witness_2}) = 1 \wedge \\ \wedge\ p_2(\mathbf{witness_2}, \mathbf{input_2}, \mathbf{witness_1}) = 1 \wedge \\ \wedge\ \mathbf{C} = \mathbf{Com(witness_2)}\}\end{aligned}$$

Let us finally define the algorithm $SNARK.PartInput$ for some srs, $state_1 \supseteq \mathbf{witness_2}$ and $\mathcal{R} = \mathcal{R}^c_{vec,com}$ as follows:

$SNARK.PartInput(srs, state_1 \supseteq \mathbf{input_1}, \mathcal{R}^c_{vec,com})$

If $c(\mathbf{input_1}) = 0$

Return

Else

Compute via interpolation on the range for $\mathscr{P}_{vec}$ *polynomials* $Q_{1,\mathbf{input_1}}(X), \ldots, Q_{m,\mathbf{input_1}}(X)$.

$\mathbf{C} = (Com(Q_{1,\mathbf{input_1}}(X)), \ldots, Com(Q_{m,\mathbf{input_1}}(X)))$

$state_2 = state_1 \cup \{\mathbf{C}\}$

$Return(state_2, \mathbf{C})$

With the above notation, **our compiler's Step 2 is:**
The alleged hybrid model SNARK $\mathscr{P}^h_{vec}$ for relation $\mathcal{R}^c_{vec,com}$ is defined as:

- *SNARK.Setup* and *SNARK.KeyGen* are the same as for relation $\mathcal{R}^c_{vec}$.
- The algorithm *SNARK.PartInput* for relation $\mathcal{R}^c_{vec}$ (see lemma 26 in appendix A) is replaced with *SNARK.PartInput* for relation $\mathcal{R}^c_{vec,com}$ as described above.
- The algorithm *SNARK.Prover* for relation $\mathcal{R}^c_{vec,com}$ is identical with the algorithm *SNARK.Prover* for relation $\mathcal{R}^c_{vec}$ (as compiled using Step 1) with the appropriate re-interpretation of the public inputs and witness.
- The algorithm *SNARK.Verifier* for relation $\mathcal{R}^c_{vec,com}$ is identical with the algorithm *SNARK.Verifier* for relation $\mathcal{R}^c_{vec}$ (as compiled using Step 1) with the appropriate re-interpretation of the public inputs and such that *SNARK.Verifier* for $\mathcal{R}^c_{vec,com}$ does not compute the polynomial commitments to the polynomials defined by the assumption (HMA.1.).

Lemma 12. *Let* $\mathscr{P}_{vec}$ *be a ranged polynomial protocol for relation* $\mathcal{R}^c_{vec}$ *defined above and let* $\mathscr{P}^*_{vec}$ *be the hybrid model SNARK for relation* $\mathcal{R}^c_{vec}$ *secure in the AGM obtained by compiling* $\mathscr{P}_{vec}$ *using our compiler's Step 1. If the hybrid model assumptions (HMA.1.) - (HMA.3.) hold w.r.t. protocol* $\mathscr{P}_{vec}$ *and* $State_{\mathcal{R}_{vec,com}} \neq \emptyset$ *then* $\mathscr{P}^h_{vec}$ *as compiled using our compiler's Step 2 is a hybrid model SNARK for relation* $\mathcal{R}^c_{vec,com}$ *secure also in the AGM.*

Proof. Let $\mathcal{E}_{KZG}$ and $\mathcal{E}$ be the extractors from the knowledge-soundness definitions for the *KZG* batch polynomial commitment scheme (as in definition 3.1, section 3 in [11]) and the hybrid model SNARK $\mathscr{P}^*_{\mathcal{R}}$ for relation $\mathcal{R}^c_{vec}$ (as per definition 5), respectively. Let $\mathcal{A}$ be an adversary against knowledge soundness in the hybrid model w.r.t. $\mathscr{P}^h_{vec}$ and relation $\mathcal{R}^c_{vec,com}$ and let $aux_{SNARK} \in \mathcal{D}$ and let $state_1 \in State_{\mathcal{R}_{vec,com}}$; let $(\mathbf{C}, state_2) = SNARK.PartInput(srs, state_1, \mathcal{R}^c_{vec,com})$. By the definition of *SNARK.PartInput* for $\mathscr{P}^h_{vec}$, there exists $\mathbf{input_1}$ such that $\mathbf{C} = \mathbf{Com}(\mathbf{input_1})$ and $c(\mathbf{input_1}) = 1$. We denote by $(\mathbf{input_2}, \pi)$ the output of $\mathcal{A}(srs, state_2, \mathcal{R}^c_{vec,com})$ and let $\mathcal{A}_1$ be the part of $\mathcal{A}$ that sends openings and batched proofs for the polynomial commitments in $\mathbf{C}$.

On the one hand, if the verifier $SNARK.Verifier(srs_{vk}, (\mathbf{C}, \mathbf{input_2}), \pi, \mathcal{R}^c_{vec,com})$ in $\mathscr{P}^h_{vec}$ accepts, then the *KZG* verifier corresponding to $\mathcal{A}_1$ also accepts. When such an event takes place, then, e.w.n.p. $\mathcal{E}_{KZG}$ extracts polynomials $Q'_1(X), \ldots, Q'_m(X)$ that represent witnesses for the vector $\mathbf{C}$ of commitments and the alleged openings of $\mathcal{A}_1$. Because the *KZG* polynomial commitment scheme is binding and by the definition of *SNARK.PartInput* for $\mathscr{P}^h_{vec}$, we obtain that $Q'_1(X) = Q_1(X), \ldots, Q'_m(X) = Q_m(X)$. Since per (HMA.3.), the set $\{Q_1(X), \ldots, Q_m(X)\}$ evaluates to $\mathbf{input_1}$ over the range over which $\mathscr{P}_{vec}$ was defined, e.w.n.p. the witness polynomials extracted by $\mathcal{E}_{KZG}$ evaluate to $\mathbf{input_1}$.

On the other hand, if the verifier $SNARK.Verifier(srs_{vk}, (\mathbf{C}, \mathbf{input_2}), \pi, \mathcal{R}^c_{vec,com})$ in $\mathscr{P}^h_{vec}$ accepts, then the verifier $SNARK.Verifier(srs_{vk}, (\mathbf{input_1}, \mathbf{input_2}), \pi, \mathcal{R}^c_{vec})$ in $\mathscr{P}^*_{vec}$ also accepts. In turn, this acceptance together with the fact that $\mathscr{P}^*_{vec}$ has knowledge-soundness as per definition 5, it implies the extractor

$\mathcal{E}$ e.w.n.p. extracts $\mathbf{witness_1}$ such that

$$(\mathbf{input_1}, \mathbf{input_2}, \mathbf{witness_1}) \in \mathcal{R}^c_{vec} \qquad (o).$$

By the definition of $SNARK.PartInput$ for $\mathscr{P}^h_{vec}$ and the way $\mathbf{input_1}$ was defined, it holds that $c(\mathbf{input_1}) = 1$. Due to (o) and by the definition of relation $\mathcal{R}^c_{vec}$, the predicates: $p_1(\mathbf{input_1}, \mathbf{input_2}, \mathbf{witness_1}) = 1$ and $p_2(\mathbf{input_1}, \mathbf{input_2}, \mathbf{witness_1}) = 1$ hold. If we let $\mathbf{witness_2} = \mathbf{input_1}$, then it is clear that

$$(\mathbf{C} = \mathbf{Com}(\mathbf{input_1}), \mathbf{input_2}, \mathbf{witness_1}, \mathbf{input_1}) \in \mathcal{R}^c_{vec,com},$$

so using $\mathcal{E}_{KZG}$ and $\mathcal{E}$ we can build an extractor for any knowledge-soundness adversary $\mathcal{A}$ for alleged hybrid model SNARK $\mathscr{P}^h_{vec}$ for relation $\mathcal{R}^c_{vec,com}$, which concludes the proof. □

To conclude this step and the detailed compiler presentation we note that it is straightforward to apply the technique described above to our SNARKs $\mathscr{P}^h_{\mathsf{ba}}$, $\mathscr{P}^h_{\mathsf{pa}}$ and $\mathscr{P}^h_{\mathsf{c}}$ compiled in Step 2 and obtain relations $\mathcal{R}^{incl}_{\mathsf{ba},com}$, $\mathcal{R}^{incl}_{\mathsf{pa},com}$ and $\mathcal{R}^{incl}_{\mathsf{c},com}$ that fulfil lemma 12.[9] For completeness, we include these three conditional NP relations below. We remind the reader that these NP relations depend on λ, but, for brevity, we have omitted it from the following definitions.

$$\mathcal{R}^{incl}_{\mathsf{ba},com} = \{(\mathbf{C} \in \mathcal{C}, \mathbf{bit} \in \mathbb{B}^n, apk \in \mathbb{F}^2; \mathbf{pk}) : apk = \sum_{i=0}^{n-2} [bit_i] \cdot pk_i \mid \mathbf{pk} \in \mathbb{G}^{n-1}_{1,inn} \wedge$$
$$\wedge \ \mathbf{C} = \mathbf{Com}(\mathbf{pk})\}$$

$$\mathcal{R}^{incl}_{\mathsf{pa},com} = \{(\mathbf{C} \in \mathcal{C}, \mathbf{b'} \in \mathbb{F}^{\frac{n}{\mathsf{block}}}_{|\mathsf{block}|}, apk \in \mathbb{F}^2; \mathbf{pk}, \mathbf{bit}) : apk = \sum_{i=0}^{n-2} [bit_i] \cdot pk_i \mid \mathbf{pk} \in \mathbb{G}^{n-1}_{1,inn} \wedge$$
$$\wedge\, \mathbf{bit} \in \mathbb{B}^n \wedge b'_j = \sum_{i=0}^{\mathsf{block}-1} 2^i \cdot bit_{\mathsf{block}\cdot j+i}, \forall j < \frac{n}{\mathsf{block}} \wedge \ \mathbf{C} = \mathbf{Com}(\mathbf{pk})\}$$

$$\mathcal{R}^{incl}_{\mathsf{c},com} = \{(\mathbf{C} \in \mathcal{C}, s \in \mathbb{F}^2, apk \in \mathbb{F}^2; \mathbf{pk}, \mathbf{bit}) : apk = \sum_{i=0}^{n-2} [bit_i] \cdot pk_i \mid \mathbf{pk} \in \mathbb{G}^{n-1}_{1,inn} \wedge$$
$$\wedge \ \mathbf{bit} \in \mathbb{B}^n \wedge \sum_{i=0}^{n-1} bit_i = s + 1 \ \wedge \mathbf{C} = \mathbf{Com}(\mathbf{pk})\}$$

3.5 Comparison between PLONK and our SNARKs

In the following, we briefly look at the differences between PLONK universal SNARK and the SNARKs designed in this work. We observe that while the NP relation that defines PLONK is more general, the relations that define our SNARKs are bespoke as we are only interested in efficiently proving public key aggregation. Because our relations are so bespoke, it turns out we do not require the full functionality that PLONK has to offer, and, in particular, our SNARKs do not require any permutation argument.

A second difference is that while PLONK's circuit is defined by a number of selector polynomials (which are a type of pre-processed polynomials) and a PLONK verifier needs to perform a one-time expensive computation of the polynomial commitments to those selector polynomials, our SNARK verifiers are able to avoid such a pre-processing phase. Indeed, in the case of $\mathscr{P}^h_{\mathsf{a}}$ (which is the only one of our three SNARKs that has a polynomial, namely $aux(X)$, that defines its circuit), our respective SNARK verifier does not need to compute a commitment to its only "selector polynomial" as, due to its structure, $aux(X)$ can be directly and efficiently evaluated by our SNARK verifier itself.

A third difference is that using our two-steps compiler, our SNARKs verifiers are able to efficiently handle input vectors of length $O(n)$, where the degree of the polynomials committed to by our SNARK provers is

[9] Note that due to our specific application and the proof-of-stake blockchain context in which we make use of our custom SNARKs, the assumption/requirement that $State_{\mathcal{R}_{vec,com}} \neq \emptyset$ for $\mathcal{R}_{vec,com} \in \{\mathcal{R}^{incl}_{\mathsf{ba},com}, \mathcal{R}^{incl}_{\mathsf{pa},com}, \mathcal{R}^{incl}_{\mathsf{c},com}\}$ is fulfilled.

also $O(n)$. Our SNARKs verifiers achieve efficiency by offloading the expensive polynomial commitment computation involving the public inputs to a trusted third party.

Moreover, while PLONK does not incorporate trusted inputs, one can easily apply the Step 2 of our compiler to PLONK. In particular, one could imagine a situation where a PLONK verifier is relying on a trusted party to compute some or all of the polynomial commitments to the circuit's selector polynomials. This is equivalent to our hybrid model SNARK definition applied to PLONK. The benefit is that by delegating such a computation, the PLONK verifier becomes more efficient.

Finally, looking ahead at our light client system instantiation in section 4.3, due to the inductive structure of the soundness proof (theorem 22), the efficiency of using a hybrid model SNARK has an even greater impact for the light client system verifier than that compared to verifying multiple instances of PLONK for the same circuit: while for the latter the PLONK verifier has to compute commitments to selector polynomial only once anyway, in the case of the former, the commitments to public inputs may differ at very step hence a trusted third party relives a higher computation burden from the light client verifier overall.

3.6 An Instantiation for Committee Key Scheme for Aggregatable Signatures

Given the SNARKs compiler described in section 3.4 and its application to the conditional NP relations mentioned at the end of that section, we are ready to present an instantiation for committee key scheme for aggregatable signatures (see section 2.3 for the definition of this notion) as used in this work (i.e. in section 4.3). We instantiate u and v introduced in section 2.3 as follows: let $u = n - 1$, where n was defined in section 2.7 and we let $v \in \mathbb{N}, n - 1 \leq v$, $v = \mathsf{poly}(\lambda)$, where by v we denote the maximum number of validators that the scheme allows.

Instantiation 13. *(Committee Key Scheme for Aggregatable Signatures) In our implementation we call committee key scheme for aggregatable signatures the following instantiation of definition 3, where* $\mathcal{R} \in \{\mathcal{R}^{incl}_{\mathsf{ba},com}, \mathcal{R}^{incl}_{\mathsf{pa},com}\}$ *as defined in the end of section 3.4:*

- $CKS_{\mathcal{R}}.Setup(v)$ *calls the following algorithms*
  $(\mathbb{G}_{1,inn}, g_{1,inn}, \mathbb{G}_{2,inn}, g_{2,inn}, \mathbb{G}_{T,inn}, e_{inn}, H_{inn}, H_{PoP}) \subset pp \leftarrow AS.Setup(aux_{AS} = v + 1)$ *which is part of instantiation 2 with the additional specification that* $aux_{AS} = v + 1$ *and using the notation detailed in section 2.2.1,*
  $srs = ([1]_{1,out}, [\tau]_{1,out}, [\tau^2]_{1,out}, \ldots, [\tau^{3v}]_{1,out}, [1]_{2,out}, [\tau]_{2,out}) \leftarrow SNARK.Setup(aux_{SNARK} = (v, 3v))$,

  $(rs_{pk}, rs_{vk}) =$
  $= (([1]_{1,out}, [\tau]_{1,out}, [\tau^2]_{1,out}, \ldots, [\tau^{3v}]_{1,out}), ([1]_{1,out}, [1]_{2,out}, [\tau]_{2,out})) \leftarrow SNARK.KeyGen(srs, \mathcal{R})$

- $ck = ([pkx]_{1,out}, [pky]_{1,out}) \leftarrow CKS_{\mathcal{R}}.GenerateCommitteeKey(rs_{pk}, (pk_i)_{i=1}^{n-1})$, *where* $\mathbf{pkx} = (pkx_1, \ldots, pkx_{n-1})$ *and* $\mathbf{pky} = (pky_1, \ldots, pky_{n-1})$ *such that* $\forall i \in \{1, \ldots, n-1\}$, $pk_i = (pkx_i, pky_i) \in \mathbb{F}^2$ *and the polynomials* $pkx(X) = \sum_{i=0}^{n-2} pkx_{i+1} \cdot L_i(X)$ *and* $pky(X) = \sum_{i=0}^{n-2} pky_{i+1} \cdot L_i(X)$ *and, finally,* $[pkx]_{1,out} = pkx(\tau) \cdot [1]_{1,out}$ *and* $[pky]_{1,out} = pky(\tau) \cdot [1]_{1,out}$.

- $\pi = (\pi_{SNARK}, apk) \leftarrow CKS_{\mathcal{R}}.Prove(rs_{pk}, ck, (pk_i)_{i=1}^{n-1}, (bit_i)_{i=1}^{n-1})$ *where* $CKS_{\mathcal{R}}.Prove$ *calls* $apk = \sum_{i=1}^{n-1} bit_i \cdot pk_i \leftarrow AS.AggregateKeys(pp, (pk_i)_{i:bit_i=1})$ *as defined in instantiation 2 and* $\pi_{SNARK} \leftarrow SNARK.Prove(rs_{pk}, (x, w), \mathcal{R})$, *for* $\mathcal{R} \in \{\mathcal{R}^{incl}_{\mathsf{ba},com}, \mathcal{R}^{incl}_{\mathsf{pa},com}\}$
  where $(x = (ck, (bit_i)_{i=1}^{n-1}||0, apk), w = (pk_i)_{i=1}^{n-1})$ *for* $\mathcal{R} = \mathcal{R}^{incl}_{\mathsf{ba},com}$ *and*
  $(x = (ck, \mathbf{b}', apk), w = ((pk_i)_{i=1}^{n-1}, (bit_i)_{i=1}^{n-1}||0)$ *for* $\mathcal{R} = \mathcal{R}^{incl}_{\mathsf{pa},com}$, *where* $\mathbf{b}'$ *is the vector of field elements formed from blocks of size* block *of bits from vector* $(bit_i)_{i=1}^{n-1}||0$ *and* block *is the highest power of 2 smaller than the size of a field element in* $\mathbb{F}$.

- $0/1 \leftarrow CKS_{\mathcal{R}}.Verify(pp, rs_{vk}, ck, m, asig, \pi, \mathbf{bitvector})$ *parses* π *to retrieve* π_{SNARK} *and* apk *and it calls* $AS.Verify(pp, apk, m, asig)$ *as defined in instantiation 2 and it also calls*

$SNARK.Verify(rs_{vk}, x, \pi_{SNARK}, \mathcal{R})$ (where π_{SNARK}, x and $\mathcal{R}$ are as defined in the paragraph above with the only difference that $(bit_i)_{i=1}^{n-1}$ represents the first $n-1$ bits of **bitvector***, padded with 0s, if not sufficiently many exist in* **bitvector***); it outputs 1 if both algorithms output 1 and it outputs 0 otherwise.*

Theorem 14. *Given the hybrid model SNARK scheme secure for relation $\mathcal{R} \in \{\mathcal{R}^{incl}_{\mathsf{ba},com}, \mathcal{R}^{incl}_{\mathsf{pa},com}\}$ as obtained using our two-step compiler in section 3.4 and the aggregatable signature scheme AS as per instantiation 2 (which fulfils definition 1) with the additional specification that $aux_{AS} = v+1$ and choosing $v = n-1$, if we assume that an efficient adversary (against the soundness of) $CKS_{\mathcal{R}}$ outputs public keys only from the source group $\mathbb{G}_{1,inn}$, then the committee key scheme $CKS_{\mathcal{R}}$ as per instantiation 13 is secure with respect to definition 3.*

For details of the proof, please see Appendix C.

4 Consensus-based Accountable Light Client Design

4.1 Problem Description: an Informal Overview

In general terms, we are interested in formalising a model in which a prover is able to convince a verifier that certain events happened in a consensus-based blockchain, while the verifier is minimally connected to the blockchain. We call such a verifier *the light client verifier* or, simply, *the light client.*

In more detail, the setting we consider is as follows. At any one time, there are at most some known maximum number of active validators. We assume that at least some threshold t' of these are honest and, hence, they are able to achieve consensus on a blockchain using a Byzantine agreement protocol. However, the validator set is not fixed forever, but it changes regularly. We call *epochs* the periods where each validator set does not change. At the end of an epoch, the current validators agree on the validator set for the next epoch. Depending on the exact details of the implementation, this operation may include checking the identity of each validator in the validator set for the next epoch and also recording those identities to the blockchain.

Given this setting, we are interested in a light client, an entity that is not part of the set of validators that allegedly work towards consensus on the blockchain, and a prover, a potentially malicious process that may not be part of any set of validators, but which listens to public messages sent by the validators. In order to model the bandwidth and/or CPU limitations of the device on which the light client runs, we assume the light client, once initialised (e.g., with some credentials/messages/events agreed upon by the first and publicly known validator set) is connected only to the prover and cannot listen to any consensus messages agreed upon by the validators. Hence, the light client completely relies on the prover to convince them that validators' consensus has been reached on a message or event that the light client is interested in.

Depending on the time frame (e.g., current epoch vs. multiple hop epoch), we can define two related problems. In the one epoch light client problem, the prover wants to convince the light client that the validator set in the current epoch agreed on something. In the multiple hop epoch light client problem, the prover needs to convince the light client that validators in a later epoch agreed on something.

Depending on the adversarial model, we have two cases as well. The first security model assumes that at least t' validators in each validator set are honest, hence there cannot be a collusion between the possibly malicious prover and the honest validators reaching consensus on the blockchain. In turn, this implies that *soundness* suffices as a security property, i.e., the prover should not be able to wrongly convince the verifier that consensus has been reached on any event or message outside of some small probability. The second security model strengthens the adversarial capability by not making any assumption regarding the fraction of honest validators in at least one of the validator sets. Hence a stronger security property is needed. We call it *accountability.* This captures the intuition that if the light client is convinced of something that the blockchain did not achieve consensus on, and if the light client and prover's communication transcript is

made public, then using it and other public information, it should be possible to identify an epoch and a number of dishonest validators equal to at least the total number of validators in that epoch minus t'.

4.2 A Formal Model for Consensus-based Accountable Light Client Design

We need the following fundamental notions:

- some number k of *epochs* with ids $1, \ldots, k$;
- for each epoch id i, $1 \leq i \leq k$, the validators on the blockchain may agree on a subset of the *set of possible consensus messages* M_i;
- associated with each consensus message m there may exist some *required data* $d_m \in D$ for some set D; when such a d_m exists, m is a binding commitment to d_m;
- a secure aggregatable signature scheme AS as defined in section 2.2.

Building on the above notions, we also define *a valid consensus view.*

Definition 15. *(Consensus view) A consensus view C for a set of epochs with ids i, $\forall i \in [k]$, for some k, contains for each epoch id i:*

- *a set PK_i of public keys (we may also consider a list of public keys and weights, e.g. proportional to stake, but we focus here on the equal weight case for simplicity).*
- *a set $\{(m, Signers, \sigma) \mid m \in M_i, Signers \subseteq PK_i\}$ where σ is a signature (or an aggregatable signature) on m and the public key(s) of the signer(s) are Signers.*
- *some required data d_m associated with each message m, such that m is a binding commitment to d_m. Note that some required data associated with different messages may overlap.*

In addition to the components mentioned above, a consensus view C contains also a genesis state genstate*;* genstate *contains the set of public keys PK_1 for the first epoch and their proofs of possession. For each of the notions contained in some epoch of C as well as for* genstate*, we say they belong to C and we simply denote that by "$\in C$".*

In the following, we assume that all algorithms processing messages use a common efficient representation that implicitly includes for each of them an epoch id; this epoch id is retrieved using a function $epoch_{id}$.

Definition 16. *(Deciding a consensus message) Given a consensus view C, we say a message $m \in M_i$ is* decided in *C if C contains valid signatures from at least some threshold t (e.g., more than 2/3) signers corresponding to public keys in PK_i or, equivalently, a valid aggregatable signature of t signers over m. Additionally, we denote by $(m, d_m) \in_{decided} C$ the fact that $m \in C$, $\exists\ d_m \in C \cap D$, d_m is the associated required data of m and m has been decided in C.*

Definition 17. *(Valid consensus view) We assume the following three* functions used for validation *are efficiently computable and they are defined as:*

- *VerifyData* $: \cup_{i=1}^{k} M_i \times D \to \{1, 0\}$ *such that it checks the validity of m given the required data d_m;*
- *HistoricVerifyData* $: \{\text{genstate}\} \times (\cup_{i=1}^{k} M_i \times D)^n \times (\cup_{i=1}^{k} PK_i)^q \to \{1, 0\}$ *such that it checks the validity of* genstate*, some set of n consensus messages and their required data and some set of q public keys;*
- *Incompatible* $: \cup_{i=1}^{k} (M_i \times M_i) \times D \to \{0, 1\}$ *which given messages m_1, m_2 and potential required data d_{m_1} for m_1 checks the incompatibility.*

Let $m_1, \ldots, m_n$ be all the distinct consensus messages contained in C. Let $pk_1, \ldots, pk_q$ be all the public keys, including repetitions, contained in $PK_i, \forall i \in [k]$. We say the consensus view C is valid *if:*

- $\exists\ d_{m_i} \in D \cap C$ *such that* $VerifyData(m_i, d_{m_i}) = 1$, $\forall 1 \leq i \leq n$.
- $HistoricVerifyData(\mathsf{genstate}, m_1, d_{m_1}, \ldots, m_n, d_{m_n}, pk_1, \ldots, pk_q) = 1$.
- *There exists no pair* (i, j), $1 \leq i, j \leq k$, $i \neq j$ *such that* $Incompatible(m_i, m_j, d_{m_i}) = 1$ *or* $Incompatible(m_j, m_i, d_{m_j}) = 1$.
- *We require that all consensus messages in* C *are decided according to definition 16.*

We conclude this subsection by defining what we mean by honest validator.

Definition 18. *(Honest validator) An honest full node of a blockchain is one that runs the protocol correctly starting from the genesis state of the blockchain. It maintains a valid consensus view of the system. A full node is a validator if they produced a public key that is in the set* PK_i *in some epoch* i *in some consensus view. An honest validator is an honest full node that is also a validator.*

4.2.1 The Case that at Least t' Validators in Each Epoch Are Honest

Next we define a light client system.

Definition 19. *(Light client system) Let* $\mathcal{R}$ *be a (conditional) NP relation. A* light client system *involves two parties -* prover *and* light client *(also called* light client verifier*) - and it implements the following algorithms:*

- $pp_{LC} \leftarrow LC.Setup(\mathcal{R})$*: a setup algorithm that takes the security parameter* λ *and a (conditional) NP relation* $\mathcal{R}$ *and outputs public parameters* pp_{LC}*.*
- $\pi \leftarrow LC.GenerateProof(pp_{LC}, C, m, \mathcal{R})$*: a proof generation algorithm that takes a valid consensus view* C*, a message* m *decided in consensus view* C *and a (conditional) NP relation* $\mathcal{R}$ *and generates a proof* π*.*
- $acc/rej \leftarrow LC.VerifyProof(pp_{LC}, LC.seed, \pi, m, \mathcal{R})$*: a proof verification algorithm that takes as input a genesis summary* $LC.seed$ *(whose properties are detailed in definition 20), a light client proof* π *and a message* m *and returns* acc *if* π *is a valid proof for* m *and* rej *otherwise.*

We call the tuple ($LC.Setup$*,* $LC.GenerateProof$*,* $LC.VerifyProof$*) a light client system if it fulfils perfect completeness and soundness as defined below.*

Perfect Completeness We say ($LC.Setup$, $LC.GenerateProof$, $LC.VerifyProof$) has perfect completeness if for any valid consensus view C and for any consensus message m decided in C we have that

$$\begin{aligned} &Pr[LC.VerifyProof(pp_{LC}, LC.seed, \pi, m, \mathcal{R}) = acc \mid pp_{LC} \leftarrow LC.Setup(\mathcal{R}),\\ &\quad \pi \leftarrow LC.GenerateProof(pp_{LC}, C, m, \mathcal{R})] = 1 \end{aligned}$$

Soundness We say ($LC.Setup$, $LC.GenerateProof$, $LC.VerifyProof$) has soundness if, for every efficient malicious prover $\mathcal{A}$,

$$\begin{aligned} &Pr[LC.VerifyProof(pp_{LC}, LC.seed, \pi, m, \mathcal{R}) = acc \mid pp_{LC} \leftarrow LC.Setup(\mathcal{R}),\\ &\quad pp \leftarrow Parse(pp_{LC}), (\pi, m, C) \leftarrow \mathcal{A}^{HonestValidator}(pp, \mathcal{R}),\\ &\quad CheckValidConsensus(C) = 1,\\ &\quad NoHonestSigning(m, OGenerateKeypair) = 1,\\ &\quad HonestThreshold(t', OGenerateKeypair, C) = 1] = negl(\lambda); \end{aligned}$$

where the predicate $CheckValidConsensus(C)$ checks if C is valid w.r.t. definition 17 and outputs 1 in that case (and 0 otherwise);

$NoHonestSigning(m, OGenerateKeypair)$ checks that there exists no public key in $Q_{keys|pk}$ (with $Q_{keys|pk}$ the restriction of Q_{keys} to the public keys and $OGenerateKeypair$ defined below) that signed

m; it outputs 1 in that case (and 0 otherwise); $\mathcal{A}^{HonestValidator}$ represents the adversary $\mathcal{A}$ interacting with honest validators. $HonestThreshold(t', OGenerateKeypair, C)$ checks that at least t' of the public keys in each PK_i of C (for every epoch i in C), are part of Q_{keys} and outputs 1 in that case (and 0 otherwise). Finally, we make the assumption that $HonestValidator$, in turn, makes oracle calls to $OGenerateKeypair(pp)$, and pp as the public parameters of aggregated signature scheme AS are part of pp_{LC}, where

$$
\begin{aligned}
&OGenerateKeypair(pp):\\
&((pk, \pi_{PoP}), sk) \leftarrow AS.GenerateKeypair(pp)\\
&Q_{keys} \leftarrow Q_{keys} \cup \{((pk, \pi_{PoP}), sk)\}\\
&\text{Output } ((pk, \pi_{PoP}), sk).
\end{aligned}
$$

Finally, we define the genesis summary and its properties with respect to a light client system.

Definition 20. *(Genesis summary) Light client verifiers have access to a genesis summary $LC.seed$, which is a well defined deterministic function of the genesis state* genstate.

4.2.2 Prover and All Validators May Collude

In the following, we extend our model above to include also the case that in at least one of the epochs less than t' of the validators are honest. We provide the definition for an *accountable light client system* which subsumes the light client system definition given above. We remark that an accountable light client system subsumes a light client system.

Definition 21. *(Accountable light client system) Let $\mathcal{R}$ be a (conditional) NP relation. An accountable light client system implements algorithms ($LC.Setup$, $LC.GenerateProof$, $LC.VerifyProof$, $LC.DetectMisbehaviour$, $LC.VerifyMisbehaviour$) where $LC.Setup$, $LC.GenerateProof$ and $LC.VerifyProof$ are defined as in 19 and*

$$(i, S, \mathbf{bit}, \sigma, m'', m') \leftarrow LC.DetectMisbehaviour(pp_{LC}, \pi, m, C, \mathcal{R})$$

is an algorithm such that it takes a proof π for message m, a consensus view C and a (conditional) NP relation $\mathcal{R}$; it outputs an epoch id i, a subset of misbehaving signers $S \subseteq PK_i$ in the same epoch as messages m'' and m', with m' decided in C and m'' signed with signature σ and using bitmask **bit** *against the set PK_i and*

$$acc/rej \leftarrow LC.VerifyMisbehaviour(pp_{LC}, i, S, \mathbf{bit}, \sigma, m'', m', C, \mathcal{R})$$

is an algorithm which takes the input of $LC.DetectMisbehaviour$ together with a consensus view C and a (conditional) NP relation $\mathcal{R}$ and checks if indeed misbehaviour took place such that completeness, soundness, accountability and accountability soundness hold, where completeness and soundness are identical to definition 19 and accountability and accountability soundness are defined below.

Accountability We say ($LC.Setup$, $LC.GenerateProof$, $LC.VerifyProof$, $LC.DetectMisbehaviour$, $LC.VerifyMisbehaviour$) achieves accountability if for every efficient adversary $\mathcal{A}$ it holds that:

$$
\begin{aligned}
Pr[&LC.VerifyMisbehaviour(pp_{LC}, LC.DetectMisbehaviour(pp_{LC}, \pi, m, C, \mathcal{R}), C, \mathcal{R}) = acc \mid\\
&pp_{LC} \leftarrow LC.Setup(\mathcal{R}), (\pi, m, C) \leftarrow \mathcal{A}(pp_{LC}, \mathcal{R}),\\
&LC.VerifyProof(pp_{LC}, LC.seed, \pi, m, \mathcal{R}) = acc, CheckValidConsensus(C) = 1,\\
&\exists\ (m', d_{m'})\ \in_{decided}\ C, Incompatible(m', m, d_{m'}) = 1, epoch_{id}(m) = epoch_{id}(m')] = 1 - negl(\lambda)
\end{aligned}
$$

Accountability Soundness We say ($LC.Setup$, $LC.GenerateProof$, $LC.VerifyProof$, $LC.DetectMisbehaviour$, $LC.VerifyMisbehaviour$) achieves accountability soundness if for every efficient

adversary $\mathcal{A}$ it holds that:

$$\begin{aligned}
&Pr[LC.VerifyMisbehaviour(pp_{LC}, i, S, \mathbf{bit}, \sigma, m'', m', C, \mathcal{R}) = acc \mid \\
&\quad (i, S, \mathbf{bit}, \sigma, m'', m', C) \leftarrow Game^{accountability-soundness}(\lambda, \mathcal{R}), \\
&\quad CheckValidConsensus(C) = 1, \\
&\quad AtLeastOneHonest(S, OGenerateKeypair) = 1] = negl(\lambda)
\end{aligned}$$

where by $\mathcal{A}^{OSpecialSign, OGenerateKeypair}$ we denote the adversary $\mathcal{A}$ having oracle access to oracles *OGenerateKeypair* as defined in section 4.2.1 and *OSpecialSign* (as defined below) and by $AtLeastOneHonest(S, OGenerateKeypair)$ we denote the predicate outputting 1 if there exists at least one public key in $S \cap Q_{keys|pk}$, where the set Q_{keys} was defined in the description of *OGenerateKeypair*; we also have the following game definition

$$\begin{aligned}
&Game^{accountability-soundness}(\lambda, \mathcal{R}) : \\
&Q_D := \emptyset \\
&Q_{keys} := \emptyset \\
&pp_{LC} \leftarrow LC.Setup(\mathcal{R}) \\
&pp \leftarrow Parse(pp_{LC}) \\
&(i, S, \mathbf{bit}, \sigma, m'', m', C) \leftarrow \mathcal{A}^{OSpecialSign, OGenerateKeypair}(pp) \\
&\text{Return } (i, S, \mathbf{bit}, \sigma, m'', m', C)
\end{aligned}$$

and

$$\begin{aligned}
&OSpecialSign(m, d_m, pk) : \\
&\text{If } VerifyData(m, d_m) = 0 \text{ then Abort} \\
&\text{If } \nexists\ \pi_{PoP}, sk \text{ s.t. } ((pk, \pi_{PoP}), sk) \in Q_{keys} \text{ then Abort} \\
&\text{For every } (m_{aux}, d_{m_{aux}}) \in Q_D \text{ s.t. } epoch_{id}(m_{aux}) = epoch_{id}(m) \\
&\quad \text{If } Incompatible(m_{aux}, m, d_{m_{aux}}) = 1 \ \vee \ Incompatible(m, m_{aux}, d_m) = 1 \text{ then Abort} \\
&\text{For } sk \text{ s.t. } ((pk, \pi_{PoP}), sk) \in Q_{keys} \\
&\quad Q_D \leftarrow Q_D \cup \{(m, d_m)\} \\
&\quad \sigma \leftarrow AS.Sign(pp, sk, m) \\
&\quad \text{Return } \sigma
\end{aligned}$$

Note that as defined above, *OSpecialSign* has read but not write access to the state of *OGenerateKeypair*. Moreover, we implicitly assume that *AS.GenerateKeypair* generates keys such that the private keys do not repeat so two users will not receive the same pair of keys (or, if they do, this happens with negligible probability). We note that, for example, for instantiation 2 this is the case.

4.3 Accountable Light Client Systems Instantiations

We motivate our light client model from 4.2 by detailing below instantiations for a light client system that is accountable light client system. Both are compatible with proof-of-stake based blockchains and, in particular, Polkadot.

4.3.1 Conventions and Assumptions

Before listing our light client systems' algorithms, we make several notational conventions:

- We use boldface font for denoting vectors. Furthermore, whenever necessary to avoid confusion, we denote by $\mathbf{Vec_i}(k)$ the k-th component of vector $\mathbf{Vec_i}$.
- In the following, unless otherwise stated, when we use $\mathcal{R}$, we mean one of the conditional relations from the set $\{\mathcal{R}^{incl}_{\mathsf{ba},com}, \mathcal{R}^{incl}_{\mathsf{pa},com}\}$.
- Given a valid consensus view C over i epochs, we assume there is a well-defined order on the set PK_j of public keys included in C, $\forall j \in [i]$; hence, in the following, we rename this set by $\mathbf{pk_j}$, $\forall j \in [i]$ and interpret it as a vector. Moreover, we instantiate honestly generated keys in $\mathbf{pk_j}$ with keys generated using $AS.GenerateKeypair$ as described in instantiation 2.
- We remind the reader that by $\mathbf{Com}(\mathbf{pk})$ we denote the set of two computationally binding polynomial commitments to the polynomials obtained by interpolating the x components of $\mathbf{pk}$ and, respectively, the y components of $\mathbf{pk}$ over a range H of size at least $v+1$, where v is some maximum number of validators that the system allows. In our instantiations for (accountable) light client systems, we use the KZG polynomial commitments, but, as mentioned also in section 3.4, the general results stated in this section hold for any binding polynomial commitments with a knowledge-soundness property.
- We assume there is a fixed upper bound v on the number of validators in each epoch and we use v in the description of our algorithms. At the same time, for compatibility with the SNARKs that we build for relations $\mathcal{R}^{incl}_{\mathsf{ba},com}$, $\mathcal{R}^{incl}_{\mathsf{pa},com}$ and $\mathcal{R}^{incl}_{\mathsf{c},com}$ as defined in 3.4, when specifically using our instantiation 13 of $CKS_{\mathcal{R}}$ or when proving our results in this section, we let v equal $n-1$, where n was defined in section 2.7.
- *Parse* and *Transform* denote functions performing the respective operations on the (accountable) light client algorithms' input in order to obtain the necessary components. *Parse* and *Transform* may additionally depend on the (conditional) relation $\mathcal{R}$ under consideration. If that is the case, we explicitly include $\mathcal{R}$. In particular, *Parse* and *Transform* functions which are part of $LC.DetectMisbehaviour$ work only for $\mathcal{R} \in \{\mathcal{R}^{incl}_{\mathsf{ba},com}, \mathcal{R}^{incl}_{\mathsf{pa},com}\}$.
- The accountable light client systems use functions f_x (deriving the public inputs), $f_{threshold}$ (deriving the Hamming weight), $HammingWeight$ (deriving the Hamming weight from consensus view elements) and f_{bit} (deriving the bitmask corresponding to public keys that signed a given message). Before providing these functions' definitions, we make the convention that, whenever used as parameters/input to these functions, $\mathbf{bit}$, apk, $\mathbf{b'}$ and s have the meaning and definition provided in section 3.

$$f_x(\mathbf{Com}(\mathbf{pk}), \mathbf{bit}, s, apk, \mathcal{R}) = \begin{cases} (\mathbf{Com}(\mathbf{pk}), \mathbf{bit}, apk) & \text{if } \mathcal{R} = \mathcal{R}^{incl}_{\mathsf{ba},com} \\ (\mathbf{Com}(\mathbf{pk}), \mathbf{b'}, apk) & \text{if } \mathcal{R} = \mathcal{R}^{incl}_{\mathsf{pa},com} \end{cases}$$

$$HammingWeight^*(\mathbf{vec}) = HammingWeight(\mathbf{vec}_1, \ldots, \mathbf{vec}_{|vec|-1})$$

$$f_{threshold}(\mathbf{x}, \mathcal{R}) = \begin{cases} HammingWeight^*(\mathbf{bit}) & \text{if } \mathcal{R} = \mathcal{R}^{incl}_{\mathsf{ba},com} \\ \sum_{j=1}^{\frac{v+1}{|block|}-1} HammingWeight(\mathbf{b'}_j) \ + \ HammingWeight^*(\mathbf{b'}_{\frac{v+1}{|block|}}) & \text{if } \mathcal{R} = \mathcal{R}^{incl}_{\mathsf{pa},com} \end{cases}$$

$$f_{bit}(C, m, v) = ((\mathbf{bit_i}(k))_{k=1}^{v} || \ 0, \sigma_{\mathbf{i}}),$$

where $i = epoch_{id}(m)$ and $\forall\ k = 1, \ldots, v$, if there exists $\sigma \in C \ \wedge \ AS.Verify(pp, \mathbf{pk_i}(k), m, \sigma) = 1$, we set $\mathbf{bit_i}(k) = 1$ and $\sigma_{\mathbf{i}}(k) = \sigma$, otherwise, we set $\mathbf{bit_i}(k) = 0$ and $\sigma_{\mathbf{i}}(k) = \_$.

Note that for each of our relations $\mathcal{R}^{incl}_{\mathsf{ba},com}$ and $\mathcal{R}^{incl}_{\mathsf{pa},com}$, apk and $\mathbf{Com}(\mathbf{pk})$ are public inputs and $\mathbf{pk}$ is a witness. Moreover, for these relations $\mathcal{R}^{incl}_{\mathsf{ba},com}$ and $\mathcal{R}^{incl}_{\mathsf{pa},com}$, we build an accountable light client system.

- We make the following instantiations: genstate is the set of public keys in $\mathbf{pk_1}$ and their alleged proofs of possession; $LC.seed = \mathbf{Com}(\mathbf{pk_1})$.

4.3.2 The Algorithms

The setup algorithm used by the accountable light client system is:

- $LC.Setup(\mathcal{R})$

$$(pp, rs_{pk}, rs_{vk}) \leftarrow CKS_{\mathcal{R}}.Setup(v)$$
$$Return\ (pp, rs_{pk}, rs_{vk})$$

The four algorithms that are part of the accountable light client system are:

- $LC.VerifyProof(pp, rs_{vk}, LC.seed, \pi, m, \mathcal{R})$

$i = epoch_{id}(m)$
$(\mathbf{\Pi}, \mathbf{\Sigma}) = Parse(\pi)$;
For $j = 1, \ldots, i$
  $(\mathbf{x_j}, \pi_{SNARK,\mathbf{j}}) = \mathbf{\Pi}(j)$; $(apk_j, com_j, \mathbf{bit_j}) = Parse(\mathbf{x_j}, \mathcal{R})$
If $LC.seed \neq com_1$
  Return rej
For $j = 1, \ldots, i$
  If $j < i$
    $m_j = (j, com_{j+1})$
  Else
    $m_j = m$
  $threshold_j = f_{threshold}(\mathbf{x_j}, \mathcal{R})$
  If $(CKS_{\mathcal{R}}.Verify(pp, rs_{vk}, com_j, m_j, \mathbf{\Sigma}(j), (\pi_{SNARK,\mathbf{j}}, apk_j), \mathbf{bit_j}) = 0) \vee (threshold_j < t)$
    Return rej
Return acc

- $LC.GenerateProof(pp, rs_{pk}, C, m, \mathcal{R})$

$\mathbf{\Pi} = ()$; $\mathbf{\Sigma} = ()$
$i = epoch_{id}(m)$
For $j = 1, \ldots, i$
  If $j < i$
    $m_j = (j, \mathbf{Com}(\mathbf{pk_{j+1}}))$
  Else
    $m_j = m$
  $(\mathbf{bit_j}, \sigma_\mathbf{j}) = f_{bit}(C, m_j, v)$
  $\mathbf{\Sigma}(j) \leftarrow AS.AggregateSignatures(pp, (\sigma_\mathbf{j}(k))_{k=1}^{v})$
  $(\pi_{SNARK,j}, apk_j) \leftarrow CKS_{\mathcal{R}}.Prove(rs_{pk}, \mathbf{Com}(\mathbf{pk_j}), (\mathbf{pk_j}(k))_{k=1}^{v}, (\mathbf{bit_j}(k))_{k=1}^{v})$
  $\mathbf{x_j} = f_x(\mathbf{Com}(\mathbf{pk_j}), \mathbf{bit_j}, s, apk_j, \mathcal{R})$
  $\mathbf{\Pi}(j) = (\mathbf{x_j}, \pi_{SNARK,j})$
Return $(\mathbf{\Pi}, \mathbf{\Sigma})$

- $LC.VerifyMisbehaviour(pp, i, S, \mathbf{bit}, \sigma, m'', m', C)$

$apk = AS.AggregateKeys(pp, (\mathbf{bit}(k) \cdot \mathbf{pk_i}(k))_{k=1}^{v})$
$(\mathbf{bit}', \_) = f_{bit}(C, m', v)$
$Compute\ S_{m''} = \{\mathbf{pk_i}(k) \mid \mathbf{bit}(k) = 1, k \in [v]\}$
$Compute\ S_{m'} = \{\mathbf{pk_i}(k) \mid \mathbf{bit}'(k) = 1, k \in [v]\}$
$If\ (AS.Verify(pp, apk, m'', \sigma) = 1)\ \wedge (S_{m''} \cap S_{m'} = S)\ \wedge\ (|S_{m'}| \geq t)\ \wedge\ (|S_{m''}| \geq t) \wedge$
$\wedge ((m', d_{m'}) \in_{decided} C) \wedge$
$\wedge\ (i = epoch_{id}(m'') = epoch_{id}(m'))\ \wedge\ (Incompatible(m'', m', d_{m'}) = 1)$
$Return\ acc$
$Else$
$Return\ rej$

- $LC.DetectMisbehaviour(pp, rs_{vk}, \pi, m, C, \mathcal{R})$

$(\mathbf{\Pi}, \mathbf{\Sigma}) = Parse(\pi)$
$i = epoch_{id}(m)$
$index = i$
$m'' = m$
$For\ j = 1, \ldots, i$
$(\mathbf{x_j}, \pi_{SNARK,j}) = \mathbf{\Pi}(j);\ (apk_j, com_j) = Parse(\mathbf{x_j})$
$If\ (LC.VerifyProof(pp, rs_{vk}, LC.seed, \pi, m, \mathcal{R}) = 1)\ \wedge\ (\exists\ min\ 2 \leq j \leq i,\ com_j \neq \mathbf{Com}(\mathbf{pk_j}))$
$m'' = (j-1, com_j);\ m' = (j-1, \mathbf{Com}(\mathbf{pk_j}))$; $index = j-1$
$ElseIf\ (LC.VerifyProof(pp, rs_{vk}, LC.seed, \pi, m, \mathcal{R}) = 1)\ \wedge\ (\forall\ 2 \leq j \leq i,\ com_j = \mathbf{Com}(\mathbf{pk_j})) \wedge$
$\wedge (\exists\ (aux, d_{aux}) \in_{decided} C)\ \wedge\ Incompatible(aux, m'', d_{aux}) = 1)$
$m' = aux$
$Else\ Return$
$\mathbf{bit} = Transform(Parse(\mathbf{x}_{index}, \mathcal{R}), \mathcal{R})$
$Compute\ S_{m''} = \{\mathbf{pk_{index}}(k) \mid \mathbf{bit}(k) = 1, k \in [v]\}$
$(\mathbf{bit}', \_) = f_{bit}(C, m', v)$
$Compute\ S_{m'} = \{\mathbf{pk_{index}}(k) \mid \mathbf{bit}'(k) = 1, k \in [v]\}$
$Return\ (index, S_{m''} \cap S_{m'}, \mathbf{bit}, \mathbf{\Sigma}(index), m'', m')$

4.3.3 Assumptions and Security Proofs

We complete our instantiation by proving the security properties of our light client and accountable light client systems according to definitions introduced in sections 4.2.1 and 4.2.2. However, beforehand, we present the assumptions we use, of which there are six classes, i.e., there are assumptions about honest validators' behaviour (B), about consensus (C), about parameters (P), about instantiation of primitives (S), about genesis state (G) and assumptions about light client integration (I).

The assumptions about honest validators' behaviour are:

- (B.1.) An honest validator never signs a message m unless it knows some required data d_m such that $VerifyData(m, d_m) = 1$ holds.
- (B.2.) An honest validator never signs a message m such that $VerifyData(m, d_m) = 1$ holds if they have previously signed m' such that $VerifyData(m', d_{m'}) = 1$ holds and $Incompatible(m, m', d_m) = 1$ or $Incompatible(m', m, d_{m'}) = 1$ hold.

- (B.3.) An honest validator does not sign any message in M_i unless they have a valid consensus view C (with $M_i \subset C$) for which their public key is in $\mathbf{pk_i}$ with $\mathbf{pk_i} \in C$.

The assumptions about consensus are:

- (C.1.) The adversary interacting with honest validators should not except with negligible probability be able to produce both: (i) a valid consensus view C in which at least t' validators in every epoch are honest that decides some message m with d_m such that $\mathit{VerifyData}(m, d_m) = 1$ and (ii) a valid consensus view C' with the same genesis state as C (in particular, with the same $\mathbf{pk_1} \subset \mathsf{genstate}$) which decides some message m' in the same epoch as m, with $\mathit{Incompatible}(m, d_m, m') = 1$.
- (C.2.) The adversary interacting with honest validators should not except with negligible probability be able to produce both: (i) a valid consensus view C in which at least t' validators in every epoch are honest and (ii) a valid consensus view C' with the same genesis state as C (in particular, with the same $\mathbf{pk_1} \subset \mathsf{genstate}$) in which there is some epoch i that C and C' both reach with $\mathbf{pk_i} \neq \mathbf{pk'_i}$.

The assumptions about parameters are:

- (P.1.) $2t - v > 0$
- (P.2.) $t + t' > v$

The assumption about instantiation of primitives is:

- (S.1.) We instantiate the aggregatable signature scheme AS such that the oracle *OSign* in definition 1 (in particular in the unforgeability property definition), is replaced with *OSpecialSign*. It is easy to see that if AS is an aggregatable signature scheme secure according to definition 1, then AS is also an aggregatable signature with oracle *OSign* replaced by *OSpecialSign* in definition 1.

The assumptions about genesis state are:

- (G.1.) In a valid consensus view, *HistoricVerifyData* checks, among others, that $\forall pk \in \mathbf{pk_1} \subset \mathsf{genstate}$, it holds that $pk \in \mathbb{G}_{1,inn}$ and that the proofs of possession for each of the public keys in $\mathbf{pk_1}$ pass the verification in $AS.\mathit{VerifyPoP}$.
- (G.2.) We assume that all honest full nodes and validators have access to the same genesis state $\mathsf{genstate}$ even when the genesis state is generated by a potential adversary.

Before the last class of assumptions, we add two notational conventions in the form of two functions:

- $\mathit{NextEpochKeys}(m, d_m)$ returns $\perp$ or a list of public keys; if $\mathit{epoch}_{id}(m) = i$, these keys are supposed to be the public keys of epoch $i + 1$.
- $\mathit{IsCommitment}(m)$ returns 0 or 1; $\mathit{IsCommitment}(m) = 1$ iff there exists some i such that $m = (i, \mathbf{Com}(\mathbf{pk_{i+1}}))$.

Finally, we make the following light client integration assumptions, i.e., these are assumptions that apply to our specific light client instantiation:

- (I.1.) If m and m' are such that $\mathit{epoch}_{id}(m) = \mathit{epoch}_{id}(m')$ and $\mathit{NextEpochKeys}(m, d_m) \neq \perp$ and $\mathit{IsCommitment}(m') = 1$ and $m' \neq (\mathit{epoch}_{id}(m), \mathbf{Com}(\mathit{NextEpochKeys}(m, d_m)))$ then

$$\mathit{Incompatible}(m, m', d_m) = 1.$$

- (I.2.) If $\mathit{epoch}_{id}(m) = i$ and $\mathit{NextEpochKeys}(m, d_m) = \mathbf{pk_{i+1}}$, then $\mathit{ValidateData}(m, d_m)$ must call $AS.\mathit{VerifyPoP}(pp, pk, \pi_{POP})$ for each $pk \in \mathbf{pk_{i+1}}$ and some data $\pi_{POP} \in d_m$ and also check that $pk \in \mathbb{G}_{1,inn}$; if any of these checks fails, then $\mathit{ValidateData}(m, d_m)$ fails.

- (I.3.) An honest validator with a valid consensus view C, does not sign a message m' with $\mathit{IsCommitment}(m') = 1$ unless there exists a message m decided in C and its required data d_m (i.e., $\mathit{ValidateData}(m, d_m) = 1$) such that

$$m' = (\mathit{epoch}_{id}(m), \mathbf{Com}(\mathit{NextEpochKeys}(m, d_m))).$$

- (I.4.) If *HistoricVerifyData* outputs 1 and there exist a message $m \in C$ that has been decided in epoch i, then for all $1 \leq j < i$, $(j, \mathbf{Com}(\mathbf{pk_{j+1}}))$ was decided in epoch j.

- (I.5.) If *HistoricVerifyData* outputs 1 and a message m' has been decided in C such that $\mathit{IsCommitment}(m') = 1$, then there exist $m, d_m \in C$ with $\mathit{ValidateData}(m, d_m) = 1$, m decided in C and $\mathit{epoch}_{id}(m) = \mathit{epoch}_{id}(m')$ such that

$$\mathbf{pk}_{\mathit{epoch}_{id}(\mathbf{m})+\mathbf{1}} = \mathit{NextEpochKeys}(m, d_m).$$

We are now ready to state and prove the security properties of our (accountable) light client systems.

Theorem 22. *If AS is the secure aggregatable signature scheme defined in instantiation 2 and if $CKS_{\mathcal{R}}$ is the secure committee key scheme defined in instantiation 13, then, together with the assumptions stated at the beginning of section 4.3.3 and for $\mathcal{R} \in \{\mathcal{R}^{incl}_{\mathsf{ba},com}, \mathcal{R}^{incl}_{\mathsf{pa},com}\}$, the tuple (LC.Setup, LC.GenerateProof, LC.VerifyProof) as instantiated in section 4.3.2 is a light client system.*

Proof. Perfect Completeness: Let m be a message decided in some epoch i of a valid consensus view C. Since C is a valid consensus view, this implies *HistoricVerifyData* outputs 1. Adding that m has been decided in epoch i and using assumption (I.4.), we have that for each previous epoch $1 \leq j < i$, $(j, \mathbf{Com}(\mathbf{pk_{j+1}}))$ was decided in epoch j; we denote this as property $(*)$. Since

$$\mathit{IsCommitment}(j, \mathbf{Com}(\mathbf{pk_{j+1}})) = 1, \forall j \in [i-1]$$

holds and using assumptions (I.5.), (I.2.) and (G.1.), we conclude the proofs of possession have been checked for each of the public keys in $\mathbf{pk_j}$, $j \in [i]$ (property $(**)$) and each of the public keys in $\mathbf{pk_j}$, $j \in [i]$ belong to $\mathbb{G}_{1,inn}$ (property $(***)$). The main fact we have to show (with the notation used in the description of *LC.VerifyProof*), is that the following two predicates hold:

$$\mathit{AS.Verify}(pp, apk_j, m_j, \mathbf{\Sigma}(j)) = 1, \forall j \in [i] \qquad (1)$$

and

$$\mathit{threshold}_j \geq t, \forall j \in [i] \qquad (2).$$

Indeed, (1) holds due to perfect completeness for aggregation for secure signature scheme instantiation *AS* which applies because: (a) for every epoch $j \in [i]$, as computed by *LC.GenerateProof*, each of the individual signatures aggregated into $\mathbf{\Sigma}(j)$ passes *AS.Verify*, (b) the aggregation $\mathbf{\Sigma}(j)$ is computed correctly as per *LC.GenerateProof*, (c) the proofs of possession have been checked for each of the public keys in $\mathbf{pk_j}$, $\forall j \in [i]$ (see property $(**)$), and, finally, (d) the aggregation of public keys denoted by apk_j, $\forall\ j \in [i]$, has been computed correctly as $(\mathbf{bit_j}(k) \cdot \mathbf{pk_j}(k))_{k=1}^{v}$ due to property $(***)$ and the perfect completeness of the SNARK scheme for relation $\mathcal{R}$ invoked by the instantiation of $CKS_{\mathcal{R}}.Prove$.
Moreover, due to definition of $f_{threshold}$ and the fact that $m_j = (j, \mathbf{Com}(\mathbf{pk_{j+1}})), \forall j \in [i-1]$ and, respectively, $m_i = m$ have been decided in their respective epochs as per $(*)$, we have that (2) holds.

Finally, using (1), letting $ck_j = com_j = \mathbf{Com}(\mathbf{pk_{j+1}})$, $\forall\ j \in [i]$ and since $\pi_{SNARK,j}$, apk_j and ck_j, $\forall\ j \in [i]$ are honestly computed as described by *LC.GenerateProof* and invoking the perfect completeness property of the $CKS_{\mathcal{R}}$ committee key scheme, we obtain that

$$CKS_{\mathcal{R}}.\mathit{Verify}(pp, rs_{vk}, \mathbf{Com}(\mathbf{pk_{j+1}}), m_j, \mathbf{\Sigma}(j), (\pi_{SNARK,j}, apk_j), \mathbf{bit_j}) = 1, \forall j \in [i] \qquad (3).$$

In turn, the fact that (2) and (3) hold with probability 1 immediately implies

$$\mathit{LC.VerifyProof}(pp_{LC}, LC.seed, \mathit{LC.GenerateProof}(pp_{LC}, C, m, \mathcal{R}), m, \mathcal{R}) = acc$$

with probability 1 (q.e.d.).

Soundness:
TO DO □

Theorem 23. *If AS is the secure aggregatable signature scheme defined in instantiation 2 and if $CKS_{\mathcal{R}}$ is the secure committee key scheme defined in instantiation 13, then, together with the assumptions stated at the beginning of section 4.3.3 and for $\mathcal{R} \in \{\mathcal{R}^{incl}_{\mathsf{ba},com}, \mathcal{R}^{incl}_{\mathsf{pa},com}\}$, the tuple (LC.Setup, LC.GenerateProof, LC.VerifyProof, LC.DetectMisbehaviour, LC.VerifyMisbehaviour) as instantiated in section 4.3.2 is an accountable light client system.*

Proof. Due to theorem 22, the tuple ($LC.Setup$, $LC.GenerateProof$, $LC.VerifyProof$, $LC.DetectMisbehaviour$, $LC.VerifyMisbehaviour$) as instantiated in section 4.3.2 is already a light client system. It is only left to show that both accountability and accountability soundness also hold.
Accountability: Let $\mathcal{A}$ be an efficient adversary that on input pp_{LC} and $\mathcal{R}$ outputs π, m and C. It is easy to see that if the descriptions of $LC.DetectMisbehaviour$ and $LC.VerifyMisbehaviour$ are followed honestly, then the predicate $S_{m''} \cap S_{m'} = S$ checked in the end of $LC.VerifyMisbehaviour$ is fulfilled. Moreover, due to the satisfied predicate

$$LC.VerifyProof(pp, rs_{vk}, LC.seed, \pi, m, \mathcal{R}) = 1 \qquad (1)$$

it holds that all $m_j, j \in [i]$ (as defined in $LC.VerifyProof$) are decided in C. Due to the way m' and m'' are computed by $LC.DetectMisbehaviour$ from the messages $(m_j)_{j=1}^{i}$, this implies $|S_{m'}| \geq t$, $|S_{m''}| \geq t$ and $(m', d_{m'}) \in_{decided} C$ and

$$Incompatible(m'', m', d_{m'}) = 1.$$

We are only left to show that

$$AS.Verify(pp, apk, m'', \sigma) = 1 \quad (*)$$

holds with overwhelming probability. Indeed, since (1) holds then, for every epoch $j \in [i]$ it holds that

$$CKS_{\mathcal{R}}.Verify(pp, rs_{vk}, com_j, m_j, \mathbf{\Sigma}(j), (\pi_{\mathbf{j}}, apk_j), \mathbf{bit_j}) = 1 \quad (2).$$

In particular, (2) holds for $j = index$. Due to soundness property of the committee key scheme $CKS_{\mathcal{R}}$, since $com_{index} = \mathbf{Com}(\mathbf{pk_{index}})$ by the definition of $index$ and $LC.DetectMisbehaviour$, since $apk_{index} = AS.AggregateKeys(pp, (\mathbf{bit_{index}}(k) \cdot \mathbf{pk_{index}}(k))_{k=1}^{v})$ as computed by $LC.DetectMisbehaviour$, since also $m'' = m_{index}$ (with $m_j, \forall j \in [i]$ defined in $LC.VerifyProof$ and $index$ defined in $LC.DetectMisbehaviour$) and, finally, since $\mathbf{\Sigma}(index) = \sigma$, as defined in $LC.DetectMisbehaviour$, it follows that $(*)$ holds with overwhelming probability (q.e.d.).

Accountability Soundness: Let $\mathcal{A}$ be an efficient adversary with oracle access to $OSpecialSign$ and $OGenerateKeypair$. If $LC.VerifyMisbehaviour(pp, i, S, \mathbf{bit}, \sigma, m'', m', C)$ outputs acc $(*)$, its checks together with completeness for aggregation imply

$$AS.Verify(pp, \sigma', m', apk_S) = 1 \quad (**),$$

where

$$\sigma_{\mathbf{i}}(j) = \begin{cases} sig & \text{if } \exists\ sig \in C, AS.Verify(pp, sig, m', \mathbf{pk_i}(j)) \\ \_ & \text{otherwise} \end{cases}$$

$$\mathbf{b_S}(j) = \begin{cases} 1 & \text{if } \mathbf{pk_i}(j) \in S \\ 0 & \text{otherwise} \end{cases}$$

$$\sigma' \leftarrow AS.AggregateSignatures(pp, (\mathbf{b_S}(j) \cdot \sigma_{\mathbf{i}}(j))_{j=1}^{v}),$$

$$apk_S \leftarrow AS.AggregateKeys(pp, (\mathbf{b_S}(j) \cdot \mathbf{pk_i}(j))_{j=1}^{v}),$$

Additionally, since $(*)$ holds and for apk as defined in $LC.VerifyMisbehaviour$, we obtain

$$AS.Verify(pp, \sigma, m'', apk) = 1 \quad (**').$$

Since $CheckValidConsensus(C) = 1$ holds and m' has been decided in epoch i of C and $d_{m'}$ is the required data associated with m', due to assumptions (I.5.), (I.4.) and (I.2.) we have that $d_{m'}$ contains correct proofs of possession for all keys in $\mathbf{pk_i}$ $(***)$.
We assume by contradiction that $AtLeastOneHonest(S, OGenerateKeypair) = 1$ holds with more than negligible probability. This is equivalent to assuming that $\exists\ pk^* \in\ S \cap Q_{keys|_{pk}}$ $(****)$ holds with more than negligible probability. Note that since the following check (which is part of $LC.VerifyMisbehaviour$) passes:

$$S_{m''} \cap S_{m'} = S,$$

any $pk \in S$ is aggregated into apk and also into apk_S; this includes pk^*. Since the aggregate signature instantiation AS is unforgeable (see definition 1 plus the assumption (S.1.)), due to $(**)$, $(**')$, $(***)$ and $(****)$ we have that, with more than negligible probability, both m' and m'' have been signed by the oracle $OSpecialSign$. However, this comes in contradiction with the fact that $Incompatible(m', m'', d_{m'}) = 1$ which is ensured as part of the checks concluding that $LC.VerifyMisbehaviour$ outputs acc. Hence, our assumption is false and $S \cap Q_{keys|_{pk}} = \emptyset$, so the probability defined in the accountability soundness property is indeed negligible.

□

Corollary 24. *In an accountable light client system, the number of misbehaving validators output by $LC.DetectMisbehaviour$ is $|S|$ and $|S| > 0$.*

Proof. Due to theorem 4.3.3, and, in particular, the accountability property, we know that given a valid consensus view C, a verifying light client proof π for a message m'' and given the existence in C of a message m' incompatible with m'', we have that the number of validators that $LC.DetectMisbehaviour$ is able to catch is at least $|S|$. Moreover, due to the accountability soundness property of an accountable light client system, we know that any public key output by $LC.DetectMisbehaviour$, e.w.n.p. , belongs to a misbehaving validator. Finally, using the accountability property and, in particular, since $LC.VerifyMisbehaviour$ accepts with overwhelming probability the output of an honest party running $LC.DetectMisbehaviour$, it holds that:

$$|S| = |S_{m'} \cap S_{m''}| = |S_{m'}| + |S_{m''}| - |S_{m'} \cup S_{m''}| \geq t + t - v > 0$$

The last inequality holds due to assumption (P.1.) and this concludes the proof. □

5 Implementation

We implemented and benchmarked the protocol. The implementation allows us to evaluate the performance of our protocol and serve as prototype for future deployment. The implementation is available at `https://github.com/w3f/apk-proofs`. It is written in Rust and uses the Arkworks library.

Table 1 gives the prover and verifier time for the three schemes (basic accountable, packed accountable and counting, see Section 3) with $v = n - 1 = 2^{10} - 1$, $v = n - 1 = 2^{16} - 1$ and $v = n - 1 = 2^{20} - 1$ signers. The benchmarks were run on commodity hardware, with an i7 ? and 16GB RAM. We remind the reader that by v we denote the maximum number of validators in our system and that n was defined in section 2.7.

These signer numbers are approximately the range of the number of validators that we were aiming our implementation at e.g. the Kusama blockchain (`https://kusama.network/`) has 1000 validators which is also the number that Polkadot is aiming for, and Ethereum 2 has about 348,000 validators and it has been suggested that there will be no more than 2^{19} (make citation out of `https://ethresear.ch/t/simplified-active-validator-cap-and-rotation-proposal/9022`).

At $v = n - 1 = 1023$, the prover can generate a proof in any scheme in well under a second, which is short enough to generate a proof for every block in most prominent blockchain protocols. Even for $v = n - 1 = 2^{20} - 1$, the prover time is under 6.4 minutes, which is the time for an Ethereum 2 epoch, the time that validators finalise the chain. For verification time, the basic accountable scheme is slower, considerably so for larger signer numbers.

Table 2 gives the number of operations the prover and verifier use. Table 3 gives the proof constituents and also the total proof and input sizes in bits. The basic accountable scheme's verifier performance at large numbers is so slow because it includes $O(n)$ field operations, which dominate the running time, however at 1023 signers it gives the smallest size. The packed accountable scheme, which includes $O(n/\lambda)$ field operations, fairs better on the benchmarks, having similar verification time than the counting scheme which has sublinear verification time, even at $2^{20} - 1$ signers. The prover is considerably slower for the latter two schemes because it needs to do additional operations. At larger signer sizes, the proof size for the accountable schemes is dominated by the bitfield.

Scheme	$v = 2^{10} - 1$		$v = 2^{16} - 1$		$v = 2^{20} - 1$	
	prover	verifier	prover	verifier	prover	verifier
Basic Accountable	587.376ms	27.388ms	20.587s	36.452ms	258.417s	146.006ms
Packed Accountable	544.614ms	16.565ms	28.398s	36.908ms	398.030s	25.867ms
Counting	510.316ms	15.758ms	28.461s	26.089ms	357.939s	27.270ms

Table 1: Proof and verifier times for the different schemes and different numbers of signers

Scheme	Prover operations	Verifier operations
Basic Accountable	$12 \times FFT(N) + FFT(4N) + 9ME(N)$	$2P + 11E + O(n)F$
Packed Accountable	$18 \times FFT(N) + FFT(4N) + 12ME(N)$	$2P + 16E + O(n/\lambda + log(n))F$
Counting	$13 \times FFT(N) + FFT(4N) + 11ME(N)$	$2P + 14E + O(log(n))F$

Table 2: Expensive prover and verifier operations. $FFT(M)$ is an FFT of size M. $ME(M)$ is a multi-exponentiation (or multi-scalar multiplication) of size M. P is a pairing, E is a single exponentiation (scalar multiplication) and F is a field operation.

Scheme	Proof	Input	Actual proof + input size in bits		
			$v = 2^{10} - 1$	$v = 2^{16} - 1$	$v = 2^{20} - 1$
Basic Accountable	$5\mathbb{G}_{1,out} + 5\mathbb{F}$	$2\mathbb{G}_{1,out} + 1\mathbb{G}_{1,inn} + n$ bits	9088	73600	1056640
Packed Accountable	$8\mathbb{G}_{1,out} + 8\mathbb{F}$	$2\mathbb{G}_{1,out} + 1\mathbb{G}_{1,inn} + n$ bits	12544	77056	1060096
Counting	$7\mathbb{G}_{1,out} + 7\mathbb{F}$	$2\mathbb{G}_{1,out} + 1\mathbb{G}_{1,inn}$	10368	10368	10368

Table 3: Proof and input constituents and total proof and input size for the implementation.

References

[1] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, and G. Wood, "Overview of polkadot and its design considerations," 2020. `https://arxiv.org/abs/2005.13456`.

[2] T. D. Team, "The internet computer for geeks," 2022. `https://internetcomputer.org/whitepaper.pdf`.

[3] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro, "Coda: Decentralized cryptocurrency at scale." Cryptology ePrint Archive, Report 2020/352, 2020.

[4] cLabs Team, "The celo protocol: A multi-asset cryptographic protocol for decentralized social payments." https://celo.org/papers/whitepaper.

[5] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains." Thesis, University of Guelph, 2016. https://atrium.lib.uoguelph.ca/xmlui/handle/10214/9769.

[6] J. Kwon and E. Buchman, "Cosmos whitepaper: A network of distributed ledgers." https://v1.cosmos.network/resources/whitepaper.

[7] J. Kwon, "Tendermint: Consensus without mining." https://tendermint.com/static/docs/tendermint.pdf.

[8] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Advances in Cryptology — ASIACRYPT 2001*, pp. 514–532, 2001.

[9] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *Advances in Cryptology - ASIACRYPT 2018*, pp. 435–464, 2018.

[10] J. Groth, "Non-interactive distributed key generation and key resharing." Cryptology ePrint Archive, Report 2021/339, 2021. https://ia.cr/2021/339.

[11] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge." Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[12] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016*, pp. 305–326, 2016.

[13] A. Gabizon, K. Gurkan, P. Jovanovic, G. Konstantopoulos, A. Oines, M. Olszewski, M. Straka, E. Tromer, and P. Vesely, "Plumo: Towards scalable interoperable blockchains using ultralight validation systems." 3rd ZKStandards Workshop, 2020. https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-plumo_celolightclient.pdf.

[14] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Aggregatable distributed key generation." Cryptology ePrint Archive, Paper 2021/005, 2021. https://eprint.iacr.org/2021/005.

[15] C. Gentry, S. Halevi, and V. Lyubashevsky, "Practical non-interactive publicly verifiable secret sharing with thousands of parties." Cryptology ePrint Archive, Paper 2021/1397, 2021. https://eprint.iacr.org/2021/1397.

[16] J. Kilian, "Uses of randomness in algorithms and protocols." PhD Thesis, 1990. https://core.ac.uk/download/pdf/4425126.pdf.

[17] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing (STOC'02)*, pp. 494–503, 2002.

[18] D. Benarroch, M. Campanelli, D. Fiore, J. Kim, J. Lee, H. Oh, and A. Querol, "Proposal: Commit-and-prove zero-knowledge proof systems and extensions," 2021. https://docs.zkproof.org/pages/standards/accepted-workshop4/proposal-commit.pdf.

[19] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology - ASIACRYPT 2010*, pp. 177–194, 2010.

[20] D. Catalano and D. Fiore, "Vector commitments and their applications," in *Public-Key Cryptography – PKC 2013*, pp. 55–72, 2013.

[21] J. Benaloh and M. de Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Advances in Cryptology — EUROCRYPT '93*, pp. 274–285, 1994.

[22] "ZKProof Community Reference. Version 0.3. Ed. by D. Benarroch, L. Brandao, M. Maller, and E. Tromer.," 2022. https://docs.zkproof.org/reference.

[23] M. Campanelli, D. Fiore, and A. Querol, "Legosnark: Modular design and composition of succinct zero-knowledge proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*, pp. 2075–2092, 2019.

[24] S. Agrawal, C. Ganesh, and P. Mohassel, "Non-interactive zero-knowledge proofs for composite statements," in *Advances in Cryptology – CRYPTO 2018*, pp. 643–673, 2018.

[25] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno, "Hash first, argue later: Adaptive verifiable computations on outsourced data," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security(CCS'16*, pp. 1304–1316, 2016.

[26] S. Yonezawa, "Pairing-friendly curves," 2020. https://tools.ietf.org/id/draft-yonezawa-pairing-friendly-curves-02.html.

[27] S. Galbraith, K. Paterson, and N. Smart, "Pairings for cryptographers." Cryptology ePrint Archive, Report 2006/165, 2006. https://ia.cr/2006/165.

[28] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Zexe: Enabling decentralized private computation," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 947–964, 2020.

[29] Y. E. Housni and A. Guillevic, "Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition." Cryptology ePrint Archive, Report 2020/351, 2020. https://eprint.iacr.org/2020/351.pdf.

[30] T. Ristenpart and S. Yilek, "The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks," in *Advances in Cryptology - EUROCRYPT 2007*, pp. 228–245, 2007.

[31] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers, "Updatable and universal common reference strings with applications to zk-snarks," in *Advances in Cryptology – CRYPTO 2018*, pp. 698–728, 2018.

[32] S. Bowe, A. Gabizon, and I. Miers, "Scalable multi-party computation for zk-snark parameters in the random beacon model." Cryptology ePrint Archive, Report 2017/1050, 2017. https://eprint.iacr.org/2017/1050.

[33] G. Fuchsbauer, E. Kiltz, and J. Loss, "The algebraic group model and its applications," in *Advances in Cryptology – CRYPTO 2018*, pp. 33–62, 2018.

[34] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *Advances in Cryptology — EUROCRYPT '97*, pp. 256–266, 1997.

[35] U. Maurer, "Abstract models of computation in cryptography," in *Cryptography and Coding*, pp. 1–12, 2005.

[36] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in *Advances in Cryptology – EUROCRYPT 2020*, pp. 738–768, 2020.

[37] K. Baghery, M. Kohlweiss, J. Siim, and M. Volkhov, "Another look at extraction and randomization of groth's zk-snark." Cryptology ePrint Archive, Report 2020/811, 2020. https://ia.cr/2020/811.

[38] N. Bitansky, R. Canetti, O. Paneth, and A. Rosen, "On the existence of extractable one-way functions," in *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing (STOC '14)*, pp. 505–514, 2014.

[39] E. Boyle and R. Pass, "Limits of extractability assumptions with distributional auxiliary input," in *Advances in Cryptology – ASIACRYPT 2015*, pp. 236–261, 2015.

[40] M. Bellare, J. A. Garay, and T. Rabin, "Fast batch verification for modular exponentiation and digital signatures." Cryptology ePrint Archive, Report 1998/007, 1998. https://eprint.iacr.org/1998/007.

[41] A. Kattis, K. Panarin, and A. Vlasov, "Redshift: Transparent snarks from list polynomial commitment iops." Cryptology ePrint Archive, Report 2019/1400, 2019. https://ia.cr/2019/1400.

[42] S. Bowe, J. Grigg, and D. Hopwood, "Recursive proof composition without a trusted setup." Cryptology ePrint Archive, Report 2019/1021, 2019. https://ia.cr/2019/1021.

[43] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*, pp. 186–194, 1987.

[44] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *Advances in Cryptology — EUROCRYPT '96*, pp. 387–398, 1996.

[45] C. Ganesh, H. Khoshakhlagh, M. Kohlweiss, A. Nitulescu, and M. Zajac, "What makes fiat–shamir zksnarks (updatable srs) simulation extractable?." Cryptology ePrint Archive, Paper 2021/511, 2021. https://eprint.iacr.org/2021/511.

A Appendix A - Ranged Polynomial Protocols for Conditional NP Relations

In the following, we keep the convention that all algorithms receive an implicit security parameter λ. The definition below is a natural extension of the notions of polynomial protocols and polynomial protocols for relations from section 4 of PLONK [11] to polynomial protocols over ranges for conditional NP relations with additional refinements required by our specific use case; these refinements are incorporated into steps 4, 5 and 6 as follows:

Definition 25. *(Polynomial Protocols over Ranges for Conditional Relations) Assume three parties, a prover $\mathcal{P}_{poly}$, a verifier $\mathcal{V}_{poly}$ and a trusted party $\mathcal{I}$. Let $\mathcal{R}^c$ be a conditional NP relation and let x be a public input both of which have been given to $\mathcal{P}_{poly}$ and $\mathcal{V}_{poly}$ by an InitGen efficient algorithm. For positive integers d, D, t, l, u, e and for set $S \subset \mathbb{F}$, an S-ranged (d, D, t, l, u, e)-polynomial protocol $\mathscr{P}_{\mathcal{R}^c}$ for relation $\mathcal{R}^c$ is a multi-round protocol between $\mathcal{P}_{poly}$, $\mathcal{V}_{poly}$ and $\mathcal{I}$ such that:*

1. *The protocol $\mathscr{P}_{\mathcal{R}^c}$ definition includes a set of pre-processed polynomials $g_1(X), \ldots, g_l(X) \in \mathbb{F}_{<d}[X]$.*
2. *The messages of $\mathcal{P}_{poly}$ are sent to $\mathcal{I}$ and are of the form $f(X)$ for $f(X) \in \mathbb{F}_{<d}[X]$.*

   If $\mathcal{P}_{poly}$ sends a message not of this form, the protocol is aborted.
3. *The messages from $\mathcal{V}_{poly}$ to $\mathcal{P}_{poly}$ are random coins.*
4. *$\mathcal{V}_{poly}$ may perform arithmetic computations using input x and the random coins used in the communication with $\mathcal{P}_{poly}$. Let $(res_1, \ldots, res_u)$ be the results of those computations which $\mathcal{V}_{poly}$ sends to $\mathcal{I}$.*
5. *Using vectors which are part of input x and/or other ad-hoc vectors which $\mathcal{V}_{poly}$ deems useful, $\mathcal{V}_{poly}$ may compute interpolation polynomials $s_1(X), \ldots, s_e(X)$ over domain S such that $s_1(X), \ldots, s_e(X) \in \mathbb{F}_{<d}[X]$. $\mathcal{V}_{poly}$ sends $s_1(X), \ldots, s_e(X)$ to $\mathcal{I}$.*
6. *At the end of the protocol, suppose $f_1(X), \ldots, f_t(X)$ are the polynomials that were sent from $\mathcal{P}_{poly}$ to $\mathcal{I}$. $\mathcal{V}_{poly}$ may ask $\mathcal{I}$ if certain polynomial identities hold between*

$$\{f_1(X), \ldots, f_t(X), g_1(X), \ldots, g_l(X), s_1(X), \ldots, s_e(X)\}$$

   over set S (i.e., if by evaluating all the polynomials that define the identity at each of the field elements from S we obtain a true statement). Each such identity is of the form

$$F(X) := G(X, h_1(v_1(X)), ..., h_M(v_M(X))) \equiv 0,$$

for some $h_i(X) \in \{f_1(X), ..., f_t(X), g_1(X), ..., g_l(X), s_1(X), \ldots, s_e(X)\}$, $G(X, X_1, \ldots, X_M) \in \mathbb{F}[X, X_1, ..., X_M]$, $v_1(X), ..., v_M(X) \in \mathbb{F}_{<d}[X]$ *such that* $F(X) \in \mathbb{F}_{<D}[X]$ *for every choice of* $f_1(X), \ldots, f_t(X)$ *made by* P_{poly} *when following the protocol correctly. Note that some of the coefficients in the identities above may be from the set* $\{res_1, \ldots, res_u\}$.

7. *After receiving the answers from* I *regarding the polynomial identities,* $\mathcal{V}_{poly}$ *outputs* acc *if all identities hold over set* S*, and outputs* rej *otherwise.*

Additionally, the following properties hold:

Perfect Completeness: *If* $\mathcal{P}_{poly}$ *follows the protocol correctly and uses a witness* ω *with* $(x, \omega) \in \mathcal{R}^c$*,* $\mathcal{V}_{poly}$ *accepts with probability one.*
Knowledge Soundness: *There exists an efficient algorithm* E*, that given access to the messages of* $\mathcal{P}_{poly}$ *to* $\mathcal{I}$ *it outputs* ω *such that, for any strategy of* $\mathcal{P}_{poly}$*, the probability of* $\mathcal{V}_{poly}$ *outputting* acc *at the end of the protocol and, simultaneously,* $(x, \omega) \in \mathcal{R}^c$ *is overwhelming in* λ.

Our definition for polynomial protocols over ranges does not include a zero-knowledge property as it is not required in our current work.

Given the definition for polynomial protocols over ranges for conditional relations as detailed above, we are now ready to state the following result. The proof follows with only minor changes from that of lemmas 4.5. and 4.7. from [11].

Lemma 26. *(Compilation of Ranged Polynomial Protocols for Conditional NP Relations into Hybrid Model SNARKs using PLONK) Let* $\mathscr{P}_{\mathcal{R}^c}$ *be a public coin* S*-ranged* (d, D, t, l, u, e)*-polynomial protocol for relation* $\mathcal{R}^c$ *where only one identity is checked by* $\mathcal{V}_{poly}$ *and predicate* c *from the definition of* $\mathcal{R}^c$ *needs to be fulfilled only by a part* x_1 *of the public input of the relation* $\mathcal{R}^c$*. Then one can construct a hybrid model SNARK protocol* $\mathscr{P}^*_{\mathcal{R}^c}$ *for relation* $\mathscr{P}_{\mathcal{R}^c}$ *with* $SNARK.PartInput$ *as defined below and with* $\mathscr{P}^*_{\mathcal{R}^c}$ *secure in the AGM under the* $2d$*-DLOG assumption*[10] *such that:*

1. *The prover* $\mathbf{P}$ *in* $\mathscr{P}^*_{\mathcal{R}^c}$ *requires* $\mathsf{e}(\mathscr{P}_{\mathcal{R}^c})$ $\mathbb{G}_{1,out}$*-exponentiations where* $\mathsf{e}(\mathscr{P}_{\mathcal{R}^c})$ *is define analogously as in PLONK (see preamble of section 4.2.), however it additionally takes into account polynomials* $s_1(X), \ldots, s_e(X)$.

2. *The total prover communication consists of* $t + t^*(\mathscr{P}_{\mathcal{R}^c}) + 1$ $\mathbb{G}_{1,out}$*-elements and* M $\mathbb{F}$*-elements, where* $t^*(\mathscr{P}_{\mathcal{R}^c})$ *is defined identically as in PLONK (see preamble of section 4.2.).*

3. *The verifier* $\mathbf{V}$ *in* $\mathscr{P}^*_{\mathcal{R}^c}$ *requires* $t + t^*(\mathscr{P}_{\mathcal{R}^c}) + 1$ $\mathbb{G}_{1,out}$*-exponentiations, two pairings and one evaluation of the polynomial* G*, and, additionally, the verifier in* $\mathscr{P}^*_{\mathcal{R}^c}$ *computes* e *polynomial commitments to polynomials in the set* $\{s_1(X), \ldots, s_e(X)\}$.

4. *The algorithm for computing partial inputs is defined as*

$$\begin{aligned}
&SNARK.PartInput(srs, state_1 \supseteq x_1, \mathcal{R}^c)\\
&If\ c(x_1) = 0\\
&\quad Return\\
&Else\\
&\quad Return(state_1, x_1)
\end{aligned}$$

[10] Definition 2.1. in PLONK [11] formally describes the $2d$-DLOG assumption.

B Appendix B - Rolled out Protocol $\mathscr{P}^h_{\mathsf{pa}}$ for Conditional NP Relation $\mathcal{R}^{incl}_{\mathsf{pa},com}$

We give below the full rolled-out hybrid model protocol $\mathscr{P}^h_{\mathsf{pa}}$ for conditional NP relation $\mathcal{R}^{incl}_{\mathsf{pa},com}$. This is obtained by applying our two-steps compiler from section 3.4 to polynomial protocol $\mathscr{P}_{\mathsf{pa}}$. In order to obtain the non-interactive version (i.e., the N from SNARK) we have additionally applied the Fiat-Shamir transform. In the following, by **transcript** at a certain point in time we denote the concatenation of the global constant, verification key, trusted public input, other public input and the proof elements created by the prover up to that point in time. $\mathcal{H}$ is a hash function, $\mathcal{H} : \{0,1\} \to \mathbb{F}$ and it emulates the random oracle. In the following, $\oplus$ is the addition operation on E_{inn} in affine coordinates. Note that in our implementation we instantiate E_{inn} with BLS12-377 [28] and E_{out} with BW6-761 [29], while we choose block to be 256 as this is the largest power of 2 smaller than the size of a field element in $\mathbb{F}$ (i.e., the base field for BLS12-377 which is the same as the scalar field of BW6-761). Finally, n has been defined as per section 2.7, i.e., n is a large enough power of 2; moreover, we let $v = n-1$ and we let $N = n$. This, in turn, ensures that N has been chosen according to the properties stated in instantiation 2, in particular when defining $AS.Setup$.

Public Parameters:
$(\mathbb{G}_{1,inn}, g_{1,inn}, \mathbb{G}_{2,inn}, g_{2,inn}, \mathbb{G}_{T,inn}, e_{inn}, H_{inn}, H_{PoP}) \subset pp \leftarrow AS.Setup(aux_{AS} = n)$

Global constant: $h \in E_{inn} \setminus \mathbb{G}_{1,inn}$

Trusted Setup: $srs \leftarrow SNARK.Setup(aux_{SNARK} = (n, 3n-3))$,
where $srs = ([1]_{1,out}, [\tau]_{1,out}, [\tau^2]_{1,out}, \ldots, [\tau^{3n-3}]_{1,out}, [1]_{2,out}, [\tau]_{2,out})$

Proving and Verifying Key Generation: $(srs_{pk}, srs_{vk}) \leftarrow SNARK.KeyGen(srs, \mathcal{R}^{incl}_{\mathsf{pa},com})$,
where $(srs_{pk}, srs_{vk}) = (([1]_{1,out}, [\tau]_{1,out}, [\tau^2]_{1,out}, \ldots, [\tau^{3n-3}]_{1,out}), ([1]_{1,out}, [1]_{2,out}, [\tau]_{2,out}))$

Partial Input: $(x_1, state_2) \leftarrow SNARK.PartInput(srs, state_1 \supseteq (pk_0, \ldots, pk_{n-2}), \mathcal{R}^{incl}_{\mathsf{pa},com})$,
where if $(pk_0, \ldots, pk_{n-2})) \notin \mathbb{G}^{n-1}_{1,inn}$, $SNARK.PartInput(srs, state_1, \mathcal{R}^{incl}_{\mathsf{pa},com})$ outputs the empty string, otherwise $SNARK.PartInput$ outputs $x_1 = ([pkx]_{1,out}, [pky]_{1,out})$ and $state_2 = state_1 \cup \{x_1\}$, where $\forall i \in \{0, \ldots, n-2\}$, pk_i as an element of the curve E_{inn} has the affine representation (pkx_i, pky_i). The polynomials $pkx(X)$ and $pky(X)$ are computed as $pkx(X) = \sum_{i=0}^{n-2} pkx_i \cdot L_i(X)$ and $pky(X) = \sum_{i=0}^{n-2} pky_i \cdot L_i(X)$ and finally, the polynomial commitments are computed as $[pkx]_{1,out} = pkx(\tau) \cdot [1]_{1,out}$ and $[pky]_{1,out} = pky(\tau) \cdot [1]_{1,out}$.

Public input: $x_1 = ([pkx]_{1,out}, [pky]_{1,out})$, $x_2 = ((b'_0, \ldots, b'_{\frac{n}{\mathsf{block}}-1}), apk)$

Witness: $w = ((pk_0, \ldots, pk_{n-2}), (bit_0, \ldots, bit_{n-1}))$

Prover's Algorithm: $\pi \leftarrow SNARK.Prove(srs_{pk}, ((x_1, x_2), w), \mathcal{R}^{incl}_{\mathsf{pa},com})$, where

Step 1:
Compute the affine representation $h = (h_x, h_y)$ and $apk \oplus h = ((apk \oplus h)_x, (apk \oplus h)_y)$.

Compute $\mathbf{pkx} = (pkx_0, \ldots, pkx_{n-2})$ and $\mathbf{pky} = (pky_0, \ldots, pky_{n-2})$ s. t. $\forall i \in \{0, \ldots, n-2\}$, pk_i as an element of the curve E_{inn} has the affine representation (pkx_i, pky_i).

Let $(kaccx_0, kaccy_0) = (h_x, h_y)$ and compute $(kaccx_{i+1}, kaccy_{i+1}) = (kaccx_i, kaccy_i) \oplus bit_i(pkx_i, pky_i)$, $\forall i < n-1$.

Compute polynomials

$$b(X) = \sum_{i=0}^{n-1} bit_i \cdot L_i(X),$$

$$kaccx(X) = \sum_{i=0}^{n-1} kaccx_i \cdot L_i(X),$$

$$kaccy(X) = \sum_{i=0}^{n-1} kaccy_i \cdot L_i(X),$$

$$pkx(X) = \sum_{i=0}^{n-2} pkx_i \cdot L_i(X),$$

$$pky(X) = \sum_{i=0}^{n-2} pky_i \cdot L_i(X).$$

Compute $[b]_{1,out} = b(\tau) \cdot [1]_{1,out}$, $[kaccx]_{1,out} = kaccx(\tau) \cdot [1]_{1,out}$, $[kaccy]_{1,out} = kaccy(\tau) \cdot [1]_{1,out}$.

The first output of the prover is $([b]_{1,out}, [kaccx]_{1,out}, [kaccy]_{1,out})$.

Step 2:
Compute the sum challenge $r = \mathcal{H}(\textbf{transcript})$.

Compute $sum = \sum_{j=0}^{\frac{n}{\mathsf{block}}-1} b'_j r^j$.

Compute: $\frac{r}{2^{\mathsf{block}-1}}, r^{\frac{n}{\mathsf{block}}}$.

Compute polynomials

$$c(X) = \sum_{i=0}^{n-1} c_i \cdot L_i(X),$$

where $c_i = 2^{i \mod \mathsf{block}} \cdot r^{i \div \mathsf{block}}, 0 \leq i \leq n-1$.

$$acc(X) = \sum_{i=0}^{n-1} acc_i \cdot L_i(X),$$

where $acc_0 = 0$ and $acc_i = \sum_{j=0}^{i-1} bit_j \cdot c_j, 0 < i \leq n-1$.

$$aux(X) = \sum_{i=0}^{n-1} aux_i \cdot L_i(X),$$

where $aux_i = 1$ if i is divisible with block and $aux_i = 0$ otherwise, $\forall i < n$

Compute $[c]_{1,out} = c(\tau) \cdot [1]_{1,out}$, $[acc]_{1,out} = acc(\tau) \cdot [1]_{1,out}$.

The second output of the prover is $([c]_{1,out}, [acc]_{1,out})$.

Step 3:
Compute the quotient challenge $\alpha = \mathcal{H}(\textbf{transcript})$.

Compute the polynomial $t(X)$ of degree at most $3 \cdot n - 3$ where

$$
\begin{aligned}
&t(X)(X^n - 1) = \\
&(X - \omega^{n-1}) \cdot [b(X) \cdot ((kaccx(X) - pkx(X))^2 \cdot (kaccx(X) + pkx(X) + kaccx(\omega \cdot X)) - (pky(X) - kaccy(X))^2) + \\
&+ (1 - b(X)) \cdot (kaccy(\omega \cdot X) - kaccy(X))] + \\
&+ \alpha(X - \omega^{n-1}) \cdot [b(X) \cdot ((kaccx(X) - pkx(X)) \cdot (kaccy(\omega \cdot X) + kaccy(X)) - (pky(X) - kaccy(X)) \cdot \\
&\cdot (kaccx(\omega \cdot X) - kaccx(X))) + (1 - b(X)) \cdot (kaccx(\omega \cdot X) - kaccx(X))] + \\
&+ \alpha^2 \cdot [b(X) \cdot (1 - b(X))] + \\
&+ \alpha^3 \cdot [c(\omega \cdot X) - c(X) \cdot (2 + (\frac{r}{2^{\mathsf{block}-1}} - 2) \cdot aux(\omega \cdot X)) - (1 - r^{\frac{n}{\mathsf{block}}}) \cdot L_{n-1}(X)] + \\
&+ \alpha^4 \cdot [(kaccx(X) - h_x) \cdot L_0(X) + (kaccx(X) - (h + apk)_x) \cdot L_{n-1}(X)] + \\
&+ \alpha^5 \cdot [(kaccy(X) - h_y) \cdot L_0(X) + (kaccy(X) - (h + apk)_y) \cdot L_{n-1}(X)] + \\
&+ \alpha^6 \cdot [acc(\omega \cdot X) - acc(X) - b(X) \cdot c(X) + sum \cdot L_{n-1}(X)] \ .
\end{aligned}
$$

Compute $[t]_{1,out} = t(\tau) \cdot [1]_{1,out}$.

The third output of the prover is $[t]_{1,out}$.

Step 4:
Compute evaluation challenge $\zeta = \mathcal{H}(\textbf{transcript})$.

Compute evaluations: $\overline{pkx} = pkx(\zeta)$, $\overline{pky} = pky(\zeta)$, $\overline{b} = b(\zeta)$, $\overline{kaccx} = kaccx(\zeta)$, $\overline{kaccy} = kaccy(\zeta)$, $\overline{c} = c(\zeta)$, $\overline{acc} = acc(\zeta)$, $\overline{t} = t(\zeta)$.

Compute linearisation polynomial:

$$
\begin{aligned}
r(X) = (\zeta - \omega^{n-1}) \cdot &[\overline{b} \cdot (\overline{kaccx} - \overline{pkx})^2 \cdot kaccx(X) + (1 - \overline{b}) \cdot kaccy(X)] + \\
&+ \alpha \cdot (\zeta - \omega^{n-1}) \cdot [\overline{b} \cdot ((\overline{kaccx} - \overline{pkx}) \cdot kaccy(X) - (\overline{pky} - \overline{kaccy}) \cdot kaccx(X)) + (1 - \overline{b}) \cdot kaccx(X)] + \\
&+ \alpha^3 \cdot c(X) + \\
&+ \alpha^6 \cdot acc(X).
\end{aligned}
$$

Compute evaluation of linearisation polynomial $\overline{r_\omega} = r(\omega \cdot \zeta)$.

The fourth output of the prover is $(\overline{pkx}, \overline{pky}, \overline{b}, \overline{kaccx}, \overline{kaccy}, \overline{c}, \overline{acc}, \overline{r_\omega})$.

Step 5:
Compute opening challenge $\nu = \mathcal{H}(\textbf{transcript})$.

Compute first opening proof polynomial

$$
\begin{aligned}
W_\zeta(X) = \frac{1}{X - \zeta} (&t(X) - \overline{t} + \\
&+ \nu(pkx(X) - \overline{pkx}) + \\
&+ \nu^2(\cdot pky(X) - \overline{pky}) + \\
&+ \nu^3(b(X) - \overline{b}) + \\
&+ \nu^4(kaccx(X) - \overline{kaccx}) + \\
&+ \nu^5(kaccy(X) - \overline{kaccy}) + \\
&+ \nu^6(c(X) - \overline{c}) + \\
&+ \nu^7(acc(X) - \overline{acc}))
\end{aligned}
$$

and second opening proof polynomial

$$W_{\zeta\cdot\omega}(X) = \frac{1}{X - \zeta\cdot\omega}(r(X) - \overline{r_\omega}).$$

Compute $[W_\zeta]_{1,out} = W_\zeta(\tau)\cdot[1]_{1,out}$ and $[W_{\zeta\cdot\omega}]_{1,out} = W_{\zeta\cdot\omega}(\tau)\cdot[1]_{1,out}$.

The fifth output of the prover is $([W_\zeta]_{1,out}, [W_{\zeta\cdot\omega}]_{1,out})$.

Compute the multipoint evaluation challenge $u = \mathcal{H}(transcript)$.

Return $\pi = ([b]_{1,out}, [kaccx]_{1,out}, [kaccy]_{1,out}, [c]_{1,out}, [acc]_{1,out}, [t]_{1,out}, [W_\zeta]_{1,out}, [W_{\zeta\cdot\omega}]_{1,out}, \overline{pkx}, \overline{pky}, \overline{b}, \overline{kaccx}, \overline{kaccy}, \overline{c}, \overline{acc}, \overline{r}_\omega)$

Verifier's Algorithm: $0/1 \leftarrow SNARK.Verify(srs_{vk}, (x_1, x_2), \pi, \mathcal{R}^{incl}_{\mathsf{pa},com})$, where

Step 1:
Compute the affine representation $h = (h_x, h_y)$ and $apk \oplus h = ((apk \oplus h)_x, (apk \oplus h)_y)$.

Step 2:
Validate proof elements $([b]_{1,out}, [kaccx]_{1,out}, [kaccy]_{1,out}, [c]_{1,out}, [acc]_{1,out}, [t]_{1,out}, [W_\zeta]_{1,out}, [W_{\zeta\cdot\omega}]_{1,out}) \in \mathbb{G}^8_{1,out}$.

Step 3:
Validate proof elements $(\overline{pkx}, \overline{pky}, \overline{b}, \overline{kaccx}, \overline{kaccy}, \overline{c}, \overline{acc}, \overline{r}_\omega) \in \mathbb{F}^8$.

Step 4:
Compute challenges $(r, \alpha, \zeta, \nu, u)$ as in the prover $P^{SNARK}_{pa,com}$ description from the common input, trusted public input, public input and respective necessary parts of the **transcript** using elements of π_{pa}.

Step 5:
Compute: $sum = \sum_{j=0}^{\frac{n}{\mathsf{block}}-1} b'_j r^j$.

Compute: $\frac{r}{2^{\mathsf{block}-1}}, r^{\frac{n}{\mathsf{block}}}$.

Step 6:
Compute polynomial evaluations $\zeta^n - 1$ and $\overline{aux}_\omega = aux(\omega\cdot\zeta)$[11] and Lagrange basis polynomials $L_0(\zeta) = \frac{\zeta^n - 1}{n\cdot(\zeta - 1)}$ and $L_{n-1}(\zeta) = \frac{(\zeta^n - 1)\cdot\omega^{n-1}}{n\cdot(\zeta - \omega^{n-1})}$.

Step 7[12]**:**
Compute quotient polynomial evaluation

$$\bar{t} = \frac{\overline{r_\omega} + [\bar{b}((\overline{kaccx} - \overline{pkx})^2\cdot(\overline{kaccx} + \overline{pkx}) - (\overline{pky} - \overline{kaccy})^2) - (1-\bar{b})\cdot\overline{kaccy}]\cdot(\zeta - \omega^{n-1})}{\zeta^n - 1} +$$

$$+\frac{\alpha\cdot[\bar{b}\cdot((\overline{kaccx} - \overline{pkx})\cdot\overline{kaccy} + (\overline{pky} - \overline{kaccy})\cdot\overline{kaccx}) - (1-\bar{b})\cdot\overline{kaccx}]\cdot(\zeta - \omega^{n-1})}{\zeta^n - 1} +$$

[11] We have $aux(\omega\cdot\zeta) = 1$ if $(\omega\cdot\zeta)^{\frac{n}{\mathsf{block}}} = 1$ and $aux(\omega\cdot\zeta) = \frac{1}{\mathsf{block}}\cdot\frac{\zeta^n - 1}{(\omega\cdot\zeta)^{\frac{n}{\mathsf{block}}} - 1}$ otherwise.

[12] This step can be optimised in obvious ways in order to reduce the number of field operations necessary to compute $\bar{t}$. We choose to include the non-compact formula in this write-up such that the reader is able to follow the linearisation process to a larger extent than via a more compact formula.

$$+\frac{\alpha^2 \cdot \bar{b} \cdot (1-\bar{b})}{\zeta^n - 1} +$$

$$-\frac{\alpha^3 \cdot [(1 - r^{\frac{n}{\mathsf{block}}}) \cdot L_{n-1}(\zeta)]}{\zeta^n - 1} - \alpha^3 \cdot \bar{c} \cdot (2 + (\frac{r}{2^{\mathsf{block}-1}} - 2)) \cdot \overline{aux}_\omega +$$

$$+\frac{\alpha^4 \cdot [(\overline{kaccx} - h_x) \cdot L_0(\zeta) + (\overline{kaccx} - (h + apk)_x) \cdot L_{n-1}(\zeta)]}{\zeta^n - 1} +$$

$$+\frac{\alpha^5 \cdot [(\overline{kaccy} - h_y) \cdot L_0(\zeta) + (\overline{kaccy} - (h + apk)_y) \cdot L_{n-1}(\zeta)]}{\zeta^n - 1} +$$

$$+\frac{\alpha^6 \cdot [-\overline{acc} - \bar{b} \cdot \bar{c} + sum \cdot L_{n-1}(\zeta)]}{\zeta^n - 1}.$$

Step 8:
Compute full batched polynomial commitment $[F]_{1,out}$.

$$\begin{aligned}
[F]_{1,out} =& [t]_{1,out} + \nu \cdot [pkx]_{1,out} + \nu^2 \cdot [pky]_{1,out} + \nu^3 \cdot [b]_{1,out} + \\
&+ (u \cdot (\zeta - \omega^{n-1}) \cdot (\bar{b} \cdot ((\overline{kaccx} - \overline{pkx})^2 + \alpha \cdot (\overline{pky} - \overline{kaccy})) + \alpha \cdot (1 - \bar{b})) + \nu^4) \cdot [kaccx]_{1,out} + \\
&+ (u \cdot (\zeta - \omega^{n-1})(\alpha \cdot \bar{b}(\overline{kaccx} - \overline{pkx}) + (1 - \bar{b})) + \nu^5) \cdot [kaccy]_{1,out} + \\
&+ (u \cdot \alpha^3 + \nu^6) \cdot [c]_{1,out} + \\
&+ (u \cdot \alpha^6 + \nu^7) \cdot [acc]_{1,out}.
\end{aligned}$$

Step 9:
Compute group-encoded batch evaluation $[E]_{1,out}$

$$[E]_{1,out} = (\bar{t} + \nu \cdot \overline{pkx} + \nu^2 \cdot \overline{pky} + \nu^3 \cdot \bar{b} + \nu^4 \cdot \overline{kaccx} + \nu^5 \cdot \overline{kaccy} + \nu^6 \cdot \bar{c} + \nu^7 \cdot \overline{acc} + u \cdot \overline{r_w}) \cdot [1]_{1,out}$$

Step 10:
Batch validate all evaluations by checking that the following holds

$$e_{out}([W_\zeta]_{1,out} + u \cdot [W_{\zeta \cdot \omega}]_{1,out}, [\tau]_{2,out}) = e_{out}(\zeta \cdot [W_\zeta]_{1,out} + u \cdot \zeta \cdot \omega \cdot [W_{\zeta \cdot \omega}]_{1,out} + [F]_{1,out} - [E]_{1,out}, [1]_{2,out}).$$

C Appendix C - Postponed Security Proof for Committee Key Scheme Instantiation

Theorem 27. *Given the hybrid model SNARK scheme secure for relation $\mathcal{R} \in \{\mathcal{R}^{incl}_{\mathsf{ba},com}, \mathcal{R}^{incl}_{\mathsf{pa},com}\}$ as obtained using our two-step compiler in section 3.4 and the aggregatable signature scheme AS as per instantiation 2 (which fulfils definition 1), with the additional specification that $aux_{AS} = v + 1$ and choosing $v = n - 1$, if we assume that an efficient adversary (against soundness of) $CKS_{\mathcal{R}}$ outputs public keys only from the source group $\mathbb{G}_{1,inn}$, then the committee key scheme $CKS_{\mathcal{R}}$ as per instantiation 13 is secure with respect to definition 3.*

Proof. We prove below the statement only for $\mathcal{R}^{incl}_{\mathsf{ba},com}$. The statement can be proven analogously for $\mathcal{R}^{incl}_{\mathsf{pa},com}$.

In order to prove perfect completeness for $CKS_{\mathcal{R}}$ instantiation 13 using a hybrid model SNARK secure for relation $\mathcal{R}^{incl}_{\mathsf{ba},com}$, we note that if $AS.Verify(pp, apk, m, asig) = 1$ holds, then due to the instantiation for $CKS_{\mathcal{R}^{incl}_{\mathsf{ba},com}}.Verify$, we have that

$$CKS_{\mathcal{R}^{incl}_{\mathsf{ba},com}}.Verify(pp, rs_{vk}, ck, m, asig, (\pi_{SNARK}, apk), (bit_i)_{i=1}^{n-1}) = 1$$

iff, in turn,

$$SNARK.Verify(rs_{vk}, (ck, (bit_i)_{i=1}^{n-1}, apk), \pi_{SNARK}, \mathcal{R}^{incl}_{\mathsf{ba},com}) = 1 \qquad (1)$$

holds. Using the fact that the keys srs and (rs_{pk}, rs_{vk}) for our hybrid model SNARK were generated correctly using $SNARK.Setup(v, 3v)$ and respectively $SNARK.KeyGen(srs, \mathcal{R}^{incl}_{\mathsf{ba},com})$, also since $(pk_i)_{i=1}^{n-1} \in \mathbb{G}^{n-1}_{1,inn}$ as honestly generated by $AS.GenerateKeypair$, then

$$(x = (ck, (bit_i)_{i=1}^{n-1}||0, apk), w = (pk_i)_{i=1}^{n-1}) \in \mathcal{R}^{incl}_{\mathsf{ba},com}$$

(because $apk = \sum_{i=1}^{n-1} bit_i \cdot pk_i$ due to instantiation 2 and ck was honestly generated as $\mathbf{Com}((pk_i)_{i=1}^{n-1})$ as a pair of binding polynomial commitments to the x and y coordinates of the keys in w, respectively) and, finally, adding that the proof π_{SNARK} was generated correctly as

$$\pi_{SNARK} \leftarrow SNARK.Prove(rs_{pk}, (x, w), \mathcal{R}^{incl}_{\mathsf{ba},com}),$$

then, by the perfect completeness property of the hybrid model SNARK for relation $\mathcal{R}^{incl}_{\mathsf{ba},com}$, we can conclude (1).

The proof for the soundness property is described below. Let $\mathcal{A}$ be an efficient adversary that, whenever it outputs a vector of public keys $(pk_i)_{i=1}^{n-1}$, the respective vector belongs to the set $\mathbb{G}^{n-1}_{1,inn}$. Assuming that the following holds

$$CKS_{\mathcal{R}^{incl}_{\mathsf{ba},com}}.Verify(pp, rs_{vk}, ck, m, asig, \pi = (\pi_{SNARK}, apk'), (bit_i)_{i=1}^{n-1}) = 1,$$

then, according to instantiation for $CKS_{\mathcal{R}^{incl}_{\mathsf{ba},com}}$, it implies that both

$$AS.Verify(pp, apk', m, asig) = 1 \qquad (2)$$

and

$$SNARK.Verify(rs_{vk}, (ck, (bit_i)_{i=1}^{n-1}||0, apk'), \pi_{SNARK}, \mathcal{R}^{incl}_{\mathsf{ba},com}) = 1 \qquad (3)$$

hold where apk' was parsed from π. Since ck was generated correctly as the pair of binding polynomial commitments $\mathbf{Com}(pk_i)_{i=1}^{n-1}$ using the vector $(pk_i)_{i=1}^{n-1}$ output by the adversary $\mathcal{A}$ (which, as per adversary definition, belongs to $\mathbb{G}^{n-1}_{1,inn}$) and due to the knowledge soundness property of the SNARK scheme secure for relation $\mathcal{R}^{incl}_{\mathsf{ba},com}$, the knowledge soundness and the computational binding property of the polynomial commitment scheme (since for our $CKS_{\mathcal{R}}$ instantiation we use the KZG commitment scheme), it implies that, with overwhelming probability $(x = (ck, (bit_i)_{i=1}^{n-1}, apk'), w = (pk_i)_{i=1}^{n-1}) \in \mathcal{R}^{incl}_{\mathsf{ba},com}$. From this, in turn, by the definition of relation $\mathcal{R}^{incl}_{\mathsf{ba},com}$, we obtain that $apk' = \sum_{i=1}^{n-1} bit_i \cdot pk_i$. Moreover, by the instantiation of aggregatable signature scheme AS, we have that $\sum_{i=1}^{n-1} bit_i \cdot pk_i = AS.AggregateKeys(pp, (pk_i)_{i:bit_i=1})$ and, as per soundness challenge definition, it holds that $apk \leftarrow AS.AggregateKeys(pp, (pk_i)_{i:bit_i=1})$. Hence $apk' = apk$. Finally, due to (2), we conclude that

$$AS.Verify(pp, apk, m, asig) = 1$$

holds with overwhelming probability (q.e.d.).

□