Inner Product Functional Commitments with Constant-Size Public Parameters and Openings

Hien Chu[1], Dario Fiore[2], Dimitris Kolonelos[2,3], and Dominique Schröder[1]

[1] Friedrich Alexander Universität Erlangen-Nürnberg, Germany
{hien.chu,dominique.schroeder}@fau.de
[2] IMDEA Software Institute, Madrid, Spain
{dario.fiore,dimitris.kolonelos}@imdea.org
[3] Universidad Politenica de Madrid, Spain

Abstract. Functional commitments (Libert et al. [ICALP'16]) allow a party to commit to a vector $\boldsymbol{v}$ of length n and later open the commitment at functions of the committed vector succinctly, namely with communication logarithmic or constant in n. Existing constructions of functional commitments rely on trusted setups and have either $O(1)$ openings and $O(n)$ parameters, or they have short parameters generatable using public randomness but have $O(\log n)$-size openings. In this work, we ask whether it is possible to construct functional commitments in which both parameters and openings can be of constant size. Our main result is the construction of the first FC schemes matching this complexity. Our constructions support the evaluation of inner products over small integers; they are built using groups of unknown order and rely on succinct protocols over these groups that are secure in the generic group and random oracle model.

1 Introduction

Commitments are one of the most fundamental cryptographic primitives having important implications in both theory and practice. In a classical commitment scheme, the sender commits to some value and hands it over to the receiver. The security of commitments guarantees that the receiver learns nothing about the committed value (hiding), while the sender cannot change the committed value afterward (binding). Commitments are one of the best-studied primitives, both in terms of underlying assumptions, integration into to more complex cryptographic schemes, and several generalizations were proposed. In this work, we study *functional commitments* (FC), proposed by Libert, Ramanna, and Yung [LRY16], that follow the goal of providing advanced functionalities while minimizing the communication complexity. In FC, the sender commits to a vector $\boldsymbol{v}$ of length n and can later open the commitment to *functions* $f(\boldsymbol{v})$ of the committed vector. A distinguishing feature of FCs is that commitment and openings should be short, namely of size logarithmic or constant in n. In terms of security, binding for FCs means that the sender cannot open the same commitment to two different outputs of the same function, i.e., to prove that both y and $y' \neq y$ are $f(\boldsymbol{v})$. Functional commitments generalize other commitments notions in prior

work that are concerned with short commitments and openings, such as vector commitments (VC) [LY10, CF13] and polynomial commitments (PC) [KZG10]. In a vector commitment, one opens single (or multiple [BBF19, LM19]) positions of the committed vector, i.e., a VC is an FC where the class of functions are the projections $f_i(\boldsymbol{v}) = v_i$. In a polynomial commitment, one commits to a polynomial $p(X)$ and opens to evaluations of p on given points z, i.e., a PC is an FC where $\boldsymbol{v}_p$ is the vector of $p(X)$'s coefficients and one opens to $f_z(\boldsymbol{v}_p) = p(z)$.

In terms of realizations, Libert et al. [LRY16] proposed an FC construction for linear functions. More recently, Lipmaa and Pavlyk [LP20] showed an FC for a class of arithmetic circuits.[4] Both these constructions [LRY16, LP20] rely on groups with pairings, and they have public parameters that must be generated in a trusted manner and whose length is at least linear in the length of the vector.

In this work, we consider the problem of realizing functional commitments that admit constant-size public parameters generated using a transparent public-coin setup. It is not hard to see that this question has a positive answer if one is willing to rely on the random oracle heuristic. In this case, one can build a functional commitment by using a succinct commitment scheme and a SNARK with transparent setup [Mic94, AHIV17, WTs+18, BCR+19, BBHR19, COS20, Set20, SL20, ZXZS20] thanks to which one can generate an opening through a SNARK proof for the NP statement that $y = f(\boldsymbol{v})$ and the commitment C opens to $\boldsymbol{v}$. However, for an NP statement of size N, all existing SNARKs with a transparent setup have proofs of length at least $O(\lambda \log N)$. Therefore, this construction implies functional commitments with logarithmic-size openings. And logarithmic-size openings is the best one can achieve in the literature even if one considers FCs for a simple functionality like inner products. Indeed, an inner product argument such as Bulletproofs [BBB+18] yields an FC for linear functions in which openings consists of $2 \cdot \log n$ elements of a group $\mathbb{G}$ where the discrete logarithm problem is hard.

To summarize, to the best of our knowledge, there is no functional commitment (including polynomial commitments) that admits constant-size and transparent public parameters and constant-size openings in the literature. The only exceptions are a few vector commitment constructions [BBF19, CFG+20] in groups of unknown order, which, however, are functional commitments for a very specific, non-algebraic, functionality. Therefore, the main question we ask in this work is:

> *Can we build functional commitments with constant-size public parameters consisting of a uniformly random string and with constant-size openings?*

1.1 Our Contributions

The main result of our paper is the construction of the first functional commitments that answer the above question in the affirmative.

[4] It is also easy to note that it is possible to construct an FC for polynomials from one for linear functions, by linearizing the polynomial.

FC for binary inner products with constant-size openings. Our first result is a functional commitment that supports the evaluation of binary inner products over the integers. Namely one can commit to a vector $\boldsymbol{v} \in \{0,1\}^n$ and, for any $\boldsymbol{f} \in \{0,1\}^n$, open the commitment to $\langle \boldsymbol{v}, \boldsymbol{f} \rangle$ computed over $\mathbb{Z}$. The scheme works over groups of unknown order and, due to the use of succinct proofs of exponentiation from [BBF19], relies on the random oracle and generic group models. The scheme's public parameters are four group elements, while openings consist of 21 elements of the hidden-order group, and 14λ bits.

While all prior FCs for inner products use techniques that somehow rely on the homomorphic property of an underlying vector commitment, our construction departs from this blueprint and shows a new set of techniques for proving an inner product. In a nutshell, we start from the first vector commitment of Campanelli et al. [CFG+20], which uses an encoding of a vector based on two RSA accumulators, and then we show how to reduce the problem of proving inner product with a public function to that of proving that a certain exponent lies in a range. To the best of our knowledge, this technique is novel. Also, a core part of this technique is a way to succinctly prove the cardinality of a set in an RSA accumulator, which we believe can be of independent interest.

FC for integer inner products. Our second result is a collection of transformations that lift an FC for binary inner products, like the one above, to one that supports the computation of inner products over the integers and over finite rings. More in detail, we show two main transformations for the following functionality: one can commit to a vector $\boldsymbol{v} \in (\mathbb{Z}_{2^\ell})^n$ and, for any $\boldsymbol{f} \in (\mathbb{Z}_{2^m})^n$, open the commitment to $\langle \boldsymbol{v}, \boldsymbol{f} \rangle$ computed over $\mathbb{Z}$.

Through the first transformation we obtain an FC whose openings have size $O(\lambda(\ell + m)) + (\ell + m)\log(\ell n)$ bits, and whose algorithms running time is approximately $(\ell + m)$ times that of the FC for binary inner products.

Through the second transformation we achieve a different tradeoff: the algorithms' running time grow by a factor $2^{\ell+m}$ but openings have a fixed size $O(\lambda)$.

We also show analogues of both transformations for the case of inner products modulo any integer p, i.e., for $\langle \cdot, \cdot \rangle : \mathbb{Z}_p^n \times \mathbb{Z}_p^n \to \mathbb{Z}_p$, that yield FCs with the same complexity as the ones above, considering ℓ and m as the bitsize of p.

Among the two, the second transformation is of particular interest because, in the case of $\ell, m = O(\log \lambda)$ (resp. $p = \mathsf{poly}(\lambda)$) it yields functional commitment schemes with constant-size openings.

If we compare to the functional commitment built using the Bulletproofs inner product argument [BBB+18], our FC has two main advantages. The first one is *flexibility*. Our FC "natively" supports inner products over $\mathbb{Z}_p$ for *any* integer p, whereas Bulletproofs only supports inner products over $\mathbb{Z}_q$ where q is the prime modulus of a group $\mathbb{G}$ where discrete logarithm holds.[5] The second advantage is that in our FC the verification algorithm admits *preprocessing*, that is, after spending $O(n)$ group exponentiations for a deterministic preprocessing

[5] One could use Bulletproofs arithmetic circuit protocol in order to simulate $\bmod p$ algebra over $\mathbb{Z}_q$, at the price of a prover's overhead.

of the function $\boldsymbol{f}$, the rest of the verification has a fixed cost $O(\lambda)$. Notably, inner product arguments based on the folding techniques of Bootle et al. [BCC$^+$16] do not admit this preprocessing, as their verification time is $O(n)$ independently of the time to read the statement.

Finally, due to the known construction of polynomial commitments from functional commitments for inner products (see above), our FCs also imply polynomial commitments with transparent setup for polynomials in $\mathbb{Z}_p[X]$.

1.2 Other related work

As mentioned earlier, functional commitments [LRY16] are related to the notion of vector commitments [LY10, CF13] and polynomial commitments [KZG10].

In a recent work, Peikert, Pepin and Sharp [PPS21] propose a new lattice-based vector commitment and also show an extension of their construction to support opening to functions, expressed as boolean circuits, of the committed vector. However, their construction works in a weaker model that assumes the availability of a trusted authority that uses a *secret key* to generate functional keys that allow one to create the opening. We argue that this is a much weaker model than the one from [LRY16] that we use in our work, where anyone can generate openings given a set of public parameters. Furthermore, the model of [PPS21] seems impossible to achieve if one aims to use public parameters generated using a public-coin setup.

Concurrent work In a very recent work Arun et al. [AGL$^+$22] also propose a functional commitment construction for inner products with transparent setup and constant-size openings, and show extensions and applications of their scheme to building constant-size SNARKs with transparent setup. Although the scheme of [AGL$^+$22] gives a solution to the same problem addressed in this paper, we emphasize that their work is fully concurrent to ours. In fact, except for sharing the fact of relying on groups of unknown order, the techniques are very different. They use extremal combinatorics techniques, whereas we rely on a new technique for proving cardinality of a set in an RSA accumulator.

2 Preliminaries

Notation. We denote by λ the security parameter. The set of all polynomial functions is denoted by $\mathsf{poly}(\lambda)(\lambda)$. We write $\epsilon(\lambda) \in \mathsf{negl}(\lambda)$ for a function $\epsilon(\lambda)$ if it vanishes faster than the inverse of any polynomial and we call it *negligible*. An algorithm $\mathcal{A}$ is said PPT if it is modeled as a probabilistic Turing machine that runs in time $\mathsf{poly}(\lambda)(\lambda)$.We use bold lowercase letters to denote a vector, e.g. $\boldsymbol{v} = (v_1, \ldots, v_n)$. With $[n]$ we denote $\{1, \ldots, n\}$ and with $[A, B]$ the set $\{A, A+1, \ldots, B-1, B\}$ where $A, B \in \mathbb{Z}, A < B$. The operator $\|\cdot\|$ is used for the bit-size, i.e. $\|x\| = \lceil \log(x) \rceil$, for $x \in \mathbb{N}$. $\mathsf{Primes}(\lambda)$ stands for the set of all primes of size λ, i.e. $\mathsf{Primes}(\lambda) = \{p : p \text{ prime } \wedge \|p\| = \lambda\}$. $O_\lambda(n)$ will mean $O(\lambda n)$ (and *not* $O(\mathsf{poly}(\lambda)n)$).

2.1 Functional Commitments

Functional commitments (FC), introduced by Libert, Ramanna and Yung [LRY16], allow a sender to commit to a vector $\boldsymbol{v}$ and then to open the commitment to a function $y = f(\boldsymbol{v})$. As in vector commitments [CF13], what makes this primitive non-trivial is a succinctness property, which requires commitments and openings to be "short", that is constant or logarithmic in the length of $\boldsymbol{v}$. In our work we use a slight generalization of the FC notion of [LRY16] considering *universal specializable public parameters*. This is a model, akin to the universal CRS of [GKM+18], where Setup creates length-independent public parameters pp, which one can later specialize to a specific length n by using a deterministic algorithm $\mathsf{Specialize}$.

Definition 1 (Functional Commitments). *A functional commitment scheme for a class of functions $\mathcal{F}$ is a tuple of algorithms $\mathsf{FC} = (\mathsf{Setup}, \mathsf{Specialize}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ with the following syntax and that satisfies correctness, succinctness, and function binding.*

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ *given the security parameter λ, outputs public parameters pp, which contain the description of a domain $\mathcal{D}$ and a universal class of functions $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$, where $\mathcal{F}_n$ is a class of n-input functions $\{f : \mathcal{D}^n \to \mathcal{R}\}$.*

$\mathsf{Specialize}(\mathsf{pp}, \mathcal{F}_n) \to \mathsf{pp}_n$ *given public parameters pp and a description of the function class $\mathcal{F}_n$, outputs specialized parameters pp_n.*

$\mathsf{Com}(\mathsf{pp}_n, \boldsymbol{v}) \to C$ *on input a vector $\boldsymbol{v} \in \mathcal{D}^n$ outputs a commitment C.*

$\mathsf{Open}(\mathsf{pp}_n, \boldsymbol{v}, f) \to \Lambda$ *on input a vector $\boldsymbol{v} \in \mathcal{D}^n$ and an admissible function $f \in \mathcal{F}_n$, outputs an opening Λ.*

$\mathsf{Ver}(\mathsf{pp}_n, C, f, y, \Lambda) \to b \in \{0,1\}$ *on input a commitment C, a function $f \in \mathcal{F}_n$, a value $y \in \mathcal{R}$, and an opening Λ, accepts ($b = 1$) or rejects ($b = 0$).*

Correctness. FC *is correct if, for any public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, any length $n \in \mathbb{N}$ and specialized $\mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$, any vector $\boldsymbol{v} \in \mathcal{D}^n$ and any admissible function $f \in \mathcal{F}_n$, it holds*

$$\mathsf{Ver}(\mathsf{pp}_n, \mathsf{Com}(\mathsf{pp}_n, \boldsymbol{v}), f, f(\boldsymbol{v}), \mathsf{Open}(\mathsf{pp}_n, \boldsymbol{v}, f)) = 1$$

Succinctness. FC *is* succinct *if there exists a fixed polynomial $p(\cdot)$ such that for any $n = \mathsf{poly}(\lambda)$, commitments and openings generated in the scheme have size at most $p(\lambda, \log n)$.*

Function binding. *For any PPT adversary $\mathcal{A}$ and any $n = \mathsf{poly}(\lambda)$, we have*

$$\Pr\begin{bmatrix} \mathsf{Ver}(\mathsf{pp}_n, C, f, y, \Lambda) = 1 & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \quad \wedge\ y \neq y' \wedge & : \ (C, f, y, \Lambda, y', \Lambda') \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathsf{Ver}(\mathsf{pp}_n, C, f, y', \Lambda') = 1 & \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, \mathcal{F}_n) \end{bmatrix} = \mathsf{negl}(\lambda)$$

Remark 1. The $\mathsf{Specialize}$ algorithm is deterministic computed from pp and $\mathcal{F}_n$. For this reason, it suffices for Function Binding that the adversary $\mathcal{A}$ takes as input pp (instead of pp_n).

Remark 2 (Preprocessing-based verification). Our FC constructions enjoy a preprocessing model of verification, similar to that of preprocessing SNARKs [GKM+18]. This means that one, given pp_n and a function f, can generate a verification key vk_f and the latter can be later used to verify any opening for f. In particular, while the cost of computing vk_f can depend on the complexity of the function, e.g., it is $O(n)$ for a linear function with n coefficients, the subsequent cost of verifying openings using vk_f depends only on the succinctness of the scheme, e.g., it is a fixed $p(\lambda)$.

2.2 Groups of unknown order

For our constructions we use groups of unknown order $\mathbb{G}$, i.e., groups where computing the order is hard. Throughout this work we will assume an efficient group sampling probabilistic algorithm $\mathsf{Ggen}(1^\lambda)$ that generates such a group $\mathbb{G}$. Potential candidates are class groups of imaginary quadratic order [BH01] and RSA groups where the factorization is unknown. The instantiation through class groups is the one that admits a public-coin (aka transparent) setup.

Hardness Assumptions. Below we recall the 2-Strong RSA assumption [BFS20], the Adaptive Root assumption [Wes19] and the Low Order assumption [BBF18].

Definition 2 (2-Strong RSA assumption [BFS20]). *We say that the* 2-strong RSA assumption *holds for* Ggen *if for any PPT adversary* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{l} u^e = g \\ \wedge e \neq 2^k,\ k \in \mathbb{N} \end{array} : \begin{array}{l} \mathbb{G} \leftarrow \mathsf{Ggen}(\lambda) \\ g \leftarrow\!\$\ \mathbb{G} \\ (u, e) \leftarrow \mathcal{A}(\mathbb{G}, g) \end{array}\right] = \mathsf{negl}(\lambda)$$

The 2-Strong RSA assumption is a special case of r-Strong RSA assumption, introduced in [BFS20]. The latter is in turn a generalization of (and trivially reduces to) the standard Strong RSA assumption [BP97] (where $r = 1$). Taking square roots can be done efficiently in Class Groups of imaginary quadratic order [BS96], but for higher order roots it is believed to be hard. Thus 2-Strong RSA assumption is believed to hold. For RSA groups the (plain) Strong RSA is a standard assumption.

Definition 3 (Adaptive Root assumption [Wes18]). *We say that the* adaptive root assumption *holds for* Ggen *if for any PPT adversary* $(\mathcal{A}_1, \mathcal{A}_2)$*:*

$$\Pr\left[\begin{array}{l} u^\ell = w \\ \wedge w \neq 1 \end{array} : \begin{array}{l} \mathbb{G} \leftarrow \mathsf{Ggen}(\lambda) \\ (w, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathbb{G}) \\ \ell \leftarrow\!\$\ \mathsf{Primes}(\lambda) \\ u \leftarrow \mathcal{A}_2(\ell, \mathsf{state}) \end{array}\right] = \mathsf{negl}(\lambda)$$

The Adaptive Root assumption is believed to hold on Class Groups and RSA groups.[6] For completeness we also recall the Low Order assumption, which is implied by the Adaptive Root assumption (see [BBF18] for the reduction).

Definition 4 (Low Order assumption [BBF18]). *We say that the* low order assumption *holds for* Ggen *if for any PPT adversary* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{l} u^\ell = 1 \\ \wedge u \neq 1 \\ \wedge 1 < \ell < 2^{\mathsf{poly}(\lambda)} \end{array} : \begin{array}{c} \mathbb{G} \leftarrow \mathsf{Ggen}(\lambda) \\ (u, \ell) \leftarrow \mathcal{A}(\mathbb{G}) \end{array}\right] = \mathsf{negl}(\lambda)$$

2.3 Arguments of Knowledge

Let $R \subset \mathcal{X} \times \mathcal{W}$ be an NP relation for a language $\mathcal{L} = \{x : \exists w \text{ s.t. } (x, w) \in R\}$. An argument system for $\mathcal{L}$ is a triple of algorithms $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ where: $\mathsf{Setup}(1^\lambda)$ takes a security parameter λ and outputs a common reference string crs; $\mathsf{P}(\mathsf{crs}, x, w)$ takes the crs, a statement x and a witness w; $\mathsf{V}(\mathsf{crs}, x)$ takes the crs, a statement x, interacts with the prover, and finally accepts or rejects. We denote an execution between the prover and verifier with $\langle \mathsf{P}(\mathsf{crs}, x, w), \mathsf{V}(\mathsf{crs}, x)\rangle = b$, with $b \in \{0, 1\}$ being the verifier's output. When V uses only public randomness, the protocol is called public coin.

Completeness. For all $(x, w) \in R$ we have

$$\Pr\left[\langle \mathsf{P}(\mathsf{crs}, x, w), \mathsf{V}(\mathsf{crs}, x)\rangle = 1 : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)\right] = 1.$$

Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be an adversary modeled as a pair of algorithms such that $\mathcal{A}_0(\mathsf{crs}) \rightarrow (x, \mathsf{state})$ (i.e. outputs an instance $x \in \mathcal{X}$ after $\mathsf{crs} \leftarrow \mathsf{Setup}(\lambda)$ is run) and $\mathcal{A}_1(\mathsf{crs}, x, \mathsf{state})$ interacts with a honest verifier. Then an argument of knowledge must satisfy the following properties:

Soundness. For all PPT $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ we have

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}_1(\mathsf{crs}, x, \mathsf{state}), \mathsf{V}(\mathsf{crs}, x)\rangle = 1 \\ \text{and } \nexists w : R(x, w) = 1 \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(\lambda) \\ (x, \mathsf{state}) \leftarrow \mathcal{A}_0(\mathsf{crs}) \end{array}\right] \in \mathsf{negl}(\lambda).$$

Knowledge Extractability. For all polynomial time adversaries $\mathcal{A}_1$ there exists a polynomial time extractor Ext such that, for all PPT $\mathcal{A}_0$ it holds

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}_1(\mathsf{crs}, x, \mathsf{state}), \mathsf{V}(\mathsf{crs}, x)\rangle = 1 \\ \text{and } (x, w') \notin \mathcal{R} \end{array} \middle| \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(\lambda) \\ (x, \mathsf{state}) \leftarrow \mathcal{A}_0(\mathsf{crs}) \\ w' \leftarrow \mathsf{Ext}(\mathsf{crs}, x, \mathsf{state}) \end{array}\right] \in \mathsf{negl}(\lambda).$$

Succinctness. Informally, an argument system is succinct if the communication and the verifier's running time in an execution of the protocol are constant or polylogarithmic in the witness length. Note that in our work, we do not need zero-knowledge arguments.

[6] In fact over the quotient group $\mathbb{G}/\{1, -1\}$ of an RSA group, since we need to exclude the element $-1 \in \mathbb{G}$ whose order is known.

2.4 Succinct Proofs of Exponentiation

We make use of the following succinct arguments of knowledge of exponents over groups of unknown order. Below we describe the protocols' functionalities and defer their description to Appendix A.

PoKE. First we recall the proof of knowledge of exponent (PoKE) of [BBF19] for the language:

$$\mathcal{L}_{\mathsf{PoKE}} = \{(Y, u; x) \in \mathbb{G} \times \mathbb{Z} : Y = u^x\}$$

parametrized by a group $\mathbb{G} \leftarrow\!\$\ \mathsf{Ggen}(\lambda)$ and a group element $g \leftarrow\!\$\ \mathbb{G}$. The protocol is succinct: it consists of 3 $\mathbb{G}$-element and 1 $\mathbb{Z}_{2^\lambda}$-element and the verification time is $O(\lambda)$, both regardless of the size, $\|x\|$, of the witness.

PoDDH. We also recall the proof of knowledge of a Diffie-Hellman tuple (PoDDH) of [CFG+20], for the language:

$$\mathcal{L}_{\mathsf{PoDDH}} = \left\{(Y_0, Y_1, Y; x_0, x_1) \in \mathbb{G}^3 \times \mathbb{Z}^2 : g_0^{x_0} = Y_0 \wedge g_1^{x_1} = Y_1 \wedge g^{x_0 x_1} = Y\right\}$$

parametrized by a group $\mathbb{G} \leftarrow\!\$\ \mathsf{Ggen}(\lambda)$ and three group elements $g, g_0, g_1 \leftarrow\!\$\ \mathbb{G}$. Notice that, unlike the usual DH-tuple, in the above protocol the bases are different and honestly generated in the setup. However, the same protocol can work for the same base, $g = g_0 = g_1$. Similarly to the PoKE, the protocol is succinct: 3 $\mathbb{G}$-elements and 2 $\mathbb{Z}_{2^\lambda}$-elements and $O(\lambda)$.

PoRE. We wil make use of a succinct protocol (PoRE) proving that the exponent of an element $Y = g^x$ lies in a certain range, $x \in [L, R]$.

$$\mathcal{L}_{\mathsf{PoRE}} = \{(Y, L, R; x) \in \mathbb{G} \times \mathbb{Z} : L < x < R \wedge g^x = Y\}\,.$$

parametrized by a group $\mathbb{G} \leftarrow\!\$\ \mathsf{Ggen}(\lambda)$ and a group element $g \leftarrow\!\$\ \mathbb{G}$.

For this we rely on the square-decomposition technique [Lip03, Gro05]. That is, an integer x is in the range $[L, R]$ if and only if there exist $(x_1, x_2, x_3) \in \mathbb{Z}^3$ such that $4(x - L)(R - x) + 1 = \sum_{i=1}^{3} x_i^2$. The proof consists of the following subprotocols (run in parallel):

- For each $i = 1, 2, 3$, the prover computes x_i, sends $Z_i = g^{x_i^2}$ for $i \in [3]$ and involves with the verifier in a succinct argument of knowledge of square exponent (PoSE) proving the validity of the last:

  $$\mathcal{L}_{\mathsf{PoSE}} = \left\{(Z_i; x_i) \in \mathbb{G} \times \mathbb{Z} : g^{x_i^2} = Z_i\right\}.$$

  PoSE is presented as a stand-alone protocol in Appendix A Figure 8.
- The prover sends $Y' = g^{(x-L)(R-x)}$ and involves in a PoDDH protocol with the verifier for the tuple (g^{x-L}, g^{R-x}, Y'). Observe that g^{x-L}, g^{R-x} can be computed homomorphically by the verifier from $Y = g^x$, thus don't have to be sent.
- Finally, the verifier merely checks if $Y'^4 \cdot g = \prod_{i=1}^{3} Z_i$.

All the above protocols are knowledge-extractable in the generic group model.

Non-interactive versions. All protocols can be made non-interactive by the standard Fiat-Shamir transformation [FS87].[7]

3 Our Functional Commitment for binary inner products

In this section we present our core construction of Functional Commitments for binary inner products with constant-size parameters and openings. Precisely, in the scheme we commit to binary vectors $\boldsymbol{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$ and the class of functions is $\mathcal{F} = \{\mathcal{F}_n\}$ where, for every positive integer n, $\mathcal{F}_n = \{f : \{0,1\}^n \to \mathbb{Z}\}$ such that f is a linear function represented as a vector of binary coefficients, i.e., $\boldsymbol{f} = (f_1, \ldots, f_n) \in \{0,1\}^n$, and computes the result as the inner product

$$y = \langle \boldsymbol{f}, \boldsymbol{v} \rangle = \sum_{i=1}^{n} f_i \cdot v_i \in \mathbb{Z}.$$

Note that, for a fixed n, every possible result y is an integer in $\{0, \ldots, n\}$. Our starting point is the vector commitment (VC) of Campanelli et. al. [CFG$^+$20], which is based on RSA accumulators [Bd94, BP97, Lip12, BBF19]. In [CFG$^+$20], each position of $\boldsymbol{v}$ is encoded as a prime, via a collision-resistant hash-to-prime function $\mathsf{Hprime}(i) \to p_i$ for each $i \in [n]$. Then, in order to commit to $\boldsymbol{v}$, one creates two RSA accumulators, C_0, C_1: the former that accumulates all primes corresponding to zero-values of $\boldsymbol{v}$ ($\{p_i = \mathsf{Hprime}(i) : v_i = 0\}$), and the latter for one-values ($\{p_i = \mathsf{Hprime}(i) : v_i = 1\}$) respectively. That is merely, $C_0 = g_0^{\prod_{v_i=0} p_i}$ and $C_1 = g_1^{\prod_{v_i=1} p_i}$. Observe that these two sets of primes form a partition of all the primes corresponding to positions $\{1, \ldots, n\}$. For binding of the commitment, they also add a succinct proof PoDDH to show that the sets in C_0 and C_1 are indeed a partition.

Starting from this vector commitment, our contribution is a new technique that allows us to create inner product opening proofs. To this end, our first key observation is that:

$$y = \langle \boldsymbol{v}, \boldsymbol{f} \rangle = \sum_i v_i f_i = \sum_{f_i=0} v_i \cdot 0 + \sum_{f_i=1} v_i \cdot 1 = \sum_{f_i=1} v_i = |\{i \in [n] : f_i = 1, v_i = 1\}|$$

since both v_i and f_i are binary. Then the prover commits to the subvector of $\boldsymbol{v}$ corresponding to positions where $f_i = 1$. This is done by using the same vector commitment described previously. That is, we compute $F_0 = g_0^{\prod_{f_i=1, v_i=0} p_i}$ and $F_1 = g_1^{\prod_{f_i=1, v_i=1} p_i}$, accompanied with a PoDDH proof π'_{PoDDH} of Diffie-Hellman tuple for the tuple (F_0, F_1, F), for $F = g^{\prod_{f_i=1} p_i}$. Notice that F can be computed by only knowing $\boldsymbol{f}$, without knowledge of $\boldsymbol{v}$.

[7] In these types of proofs though one should set ℓ to be of size 2λ for the non-interactive case [BBF18].

The next step to prove the inner product is to show that (F_0, F_1) is actually a commitment to a subvector of $\boldsymbol{v}$. This is done by showing that F_0 accumulates a subset of the primes of C_0, and similarly F_1 accumulates a subset of the primes of C_1. Putting it in other words, the 'exponent' of F_b is contained in the accumulator C_b: there is a W_b such that $W_b^{\prod_{f_i=1, v_i=b} p_i} = C_b$ and $F_b = g_b^{\prod_{f_i=1, v_i=b} p_i}$, for $b = 0, 1$. The last can be proven in a succinct way via a simple concatenation of two PoKE proofs, π_0, π_1.

Observe now that the F_1 accumulates exactly the (primes corresponding to) positions that contribute to the inner product $\{i \in [n] : f_i = 1, v_i = 1\}$. The number of primes that F_1 contains in its 'exponent' is exactly y. All that is missing now is a way to convince the verifier about the number of primes in the exponent of F_1. For this, we set the size of each prime p_i to be such that the range of any product $\prod_{i \in \mathcal{I}} p_i$ determines uniquely the cardinality of $\mathcal{I}$ (i.e., the number of primes in the product). This way, a range proof for the 'exponent' of F_1 can convince the verifier about the cardinality of the accumulated set, which is the inner product result y. For this, we generate a succinct range proof π_{PoRE} using our protocol of section 2.4.

The verifier, holding the commitment $(C_0, C_1, \pi_{\mathsf{PoDDH}})$, receives the opening proof $(F_0, F_1, \pi'_{\mathsf{PoDDH}}, W_0, \pi_0, W_1, \pi_1, \pi_{\mathsf{PoRE}})$. It is important to make sure that F_1 contains exactly the primes of positions where $f_i = 1$ and $v_i = 1$. The (F_1, C_1)-'subvector' proof π_1 ensures that $v_i = 1$ for all its primes (since C_1 contains only primes for $v_i = 1$). For the $f_i = 1$ part, the verifier herself computes $F = g^{\prod_{f_i=1} p_i}$ and verifies that (F_0, F_1, F) is a DH tuple through π'_{PoDDH}. This ensures that (1) all the primes in the exponents of F_0, F_1 are for $f_i = 1$ and (2) no position i for $f_i = 1$ was excluded maliciously; all of them were either put in F_0 or F_1. This convinces the verifier that exactly the positions i where $f_i = 1, v_i = 1$ are in the 'exponent' of F_1.

3.1 Functional VCs for binary linear functions from range proofs

Here we formally describe our construction. We simplify the notation omitting the indicator $i \in [n]$ from the sums and the products below. For example $\sum_i x_i$ would implicitly mean $\sum_{i \in [n]} x_i$ and $\prod_{v_i=1} x_i$ would implicitly mean $\prod_{i \in [n], v_i=1} x_i$. Furthermore, we use abbreviations for some products we will use that can be found in Figure 1.

$\mathsf{prod} = \prod_i \mathsf{Hprime}(i)$	$\mathsf{fprod} = \prod_{f_i=1} \mathsf{Hprime}(i)$
$\mathsf{prod}_0 = \prod_{v_i=0} \mathsf{Hprime}(i)$	$\mathsf{fprod}_0 = \prod_{f_i=1, v_i=0} \mathsf{Hprime}(i)$
$\mathsf{prod}_1 = \prod_{v_i=1} \mathsf{Hprime}(i)$	$\mathsf{fprod}_1 = \prod_{f_i=1, v_i=1} \mathsf{Hprime}(i)$

Fig. 1: Summary of symbols for the products used in the construction.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: The setup algorithm generates a hidden order group $\mathbb{G} \leftarrow \mathsf{Ggen}(1^\lambda)$ and samples three generators $g, g_0, g_1 \leftarrow\!\!\$\, \mathbb{G}$. It determines a collision-resistant function Hprime that maps integers to primes and it returns $\mathsf{pp} = (\mathbb{G}, g, g_0, g_1)$.

$\mathsf{Specialize}(\mathsf{pp}, n) \to \mathsf{pp}_n$: The algorithm samples a collision-resistant function Hprime that maps integers to primes.[8] Computes $\mathsf{prod} = \prod_i \mathsf{Hprime}(i)$ and sets $U_n = g^{\mathsf{prod}}$. Returns $\mathsf{pp}_n = (\mathsf{pp}, \mathsf{Hprime}, U_n)$.

$\mathsf{Com}(\mathsf{pp}_n, \boldsymbol{v}) \to C$: The commitment algorithm takes as input a vector of bits $\boldsymbol{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$. It computes the product of all primes that correspond to a zero-value of the vector (i.e., $v_i = 0$) as $\mathsf{prod}_0 = \prod_{v_i=0} \mathsf{Hprime}(i)$, and similarly $\mathsf{prod}_1 = \prod_{v_i=1} \mathsf{Hprime}(i)$ for the one-values. Next, it computes the accumulators

$$C_0 = g_0^{\mathsf{prod}_0} \qquad \text{and} \qquad C_1 = g_1^{\mathsf{prod}_1}$$

and a PoDDH proof $\pi = \mathsf{PoDDH.P}\left((\mathbb{G}, g, g_0, g_1), (C_0, C_1, U_n), (\mathsf{prod}_0, \mathsf{prod}_1)\right)$, which ensures that, given the above (C_0, C_1, U_n), it holds $\mathsf{prod} = \mathsf{prod}_0 \cdot \mathsf{prod}_1$:

$$\mathcal{L} = \left\{ (C_0, C_1, U_n; \mathsf{prod}_0, \mathsf{prod}_1) : g_0^{\mathsf{prod}_0} = C_0 \wedge g_1^{\mathsf{prod}_1} = C_1 \wedge g^{\mathsf{prod}_0 \cdot \mathsf{prod}_1} = U_n \right\}$$

Returns $C = (C_0, C_1, \pi)$.

$\mathsf{Open}(\mathsf{pp}_n, C, \boldsymbol{v}, \boldsymbol{f}) \to (y, \Lambda)$: $\boldsymbol{f} = (f_1, \ldots, f_n) \in \{0,1\}^n$ is a vector of bits. The output of the function is

$$y = \langle \boldsymbol{v}, \boldsymbol{f} \rangle = \sum v_i f_i = \sum_{f_i=0} v_i \cdot 0 + \sum_{f_i=1} v_i \cdot 1 = \sum_{f_i=1} v_i$$

Let $\mathsf{fprod} = \prod_{f_i=1} \mathsf{Hprime}(i)$, $\mathsf{fprod}_0 = \prod_{f_i=1, v_i=0} \mathsf{Hprime}(i)$ and $\mathsf{fprod}_1 = \prod_{f_i=1, v_i=1} \mathsf{Hprime}(i)$. Computes $F = g^{\mathsf{fprod}}$ and

$$F_0 = g_0^{\mathsf{fprod}_0} \qquad \text{and} \qquad F_1 = g_1^{\mathsf{fprod}_1}$$

Then computes the following arguments of knowledge:

- π_0: a proof that F_0 contains a 'subvector' of C_0, i.e. a proof for the language:

$$\mathcal{L}_0 = \left\{ (F_0, W_0; \mathsf{fprod}_0) : W_0^{\mathsf{fprod}_0} = C_0 \wedge g_0^{\mathsf{fprod}_0} = F_0 \right\}$$

- π_1: a proof that F_1 contains a 'subvector' of C_1, i.e. a proof for the language:

$$\mathcal{L}_1 = \left\{ (F_1, W_1; \mathsf{fprod}_1) : W_1^{\mathsf{fprod}_1} = C_1 \wedge g_1^{\mathsf{fprod}_1} = F_1 \right\}$$

- π_2: a PoDDH for F_0, F_1, F:

$$\mathcal{L}_2 = \left\{ (F_0, F_1, F; \mathsf{fprod}_0, \mathsf{fprod}_1) : g_0^{\mathsf{fprod}_0} = F_0 \wedge g_1^{\mathsf{fprod}_1} = F_1 \wedge g^{\mathsf{fprod}_0 \cdot \mathsf{fprod}_1} = F \right\}$$

- π_3: a range proof that fprod_1 is in a certain range $L(y) < \mathsf{fprod}_1 < R(y)$, that is uniquely determined by y. L and R are public functions that depend on Hprime (see Figure 2 for their concrete description).

$$\mathcal{L}_3 = \left\{ (F_1, y; \mathsf{fprod}_1) : L(y) < \mathsf{fprod}_1 < R(y) \wedge g_1^{\mathsf{fprod}_1} = F_1 \right\}$$

[8] As we discuss next and in more detail in Section 3.3, the choice of Hprime depends on n.

Returns $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$

$\mathsf{Ver}(\mathsf{pp}_n, C, \Lambda, \boldsymbol{f}, y) \to b$: It computes $F = g^{\mathsf{fprod}} = g^{\prod_{f_i=1} \mathsf{Hprime}(i)}$ that depends only on $\boldsymbol{f}$ and outputs 1 iff all $\pi, \pi_0, \pi_1, \pi_2, \pi_3$ verify. Notice that computing F is necessary as is an input to the proof π_2.

$\mathsf{Hprime} : [n] \to \left(2^{\kappa(\lambda)}, 2^{\kappa(\lambda)+\frac{\kappa(\lambda)}{n}}\right)$ collision-resistant hash-to-prime function.

$$L(y) = \begin{cases} 2^{\kappa(\lambda)y}, & y \in [n] \\ 1, & y = 0 \end{cases} \quad \text{and} \quad R(y) = \begin{cases} 2^{(\kappa(\lambda)+\frac{\kappa(\lambda)}{n})y}, & y \in [n] \\ 1, & y = 0 \end{cases}$$

Fig. 2: Definitions of the range functions L, R. The functions depend on the range Hprime, which in turn depends on n and λ (specified in the setup and specialize phases respectively).

Remark 3. For ease of presentation, in the Open algorithm, we describe four distinct proofs, $\pi_0, \pi_1, \pi_2, \pi_3$. In order to optimize the proof size, they can be merged into a single proof avoiding redundancies. We present in details the (merged) protocol in Section 3.3.

Determining the hash function and the range We need to find a proper hash-to-prime function and a corresponding range $[L(y), R(y)]$ for fprod_1 such that for any $y = 1, \ldots, n$:

$$\mathsf{fprod}_1 := \prod_{v_i=1, f_i=1} \mathsf{Hprime}(i) \in [L(y), R(y)] \iff \left|\{i \in [n] : v_i = 1, f_i = 1\}\right| = y$$

meaning that a range for the product of the primes should translate to its number of prime factors. And the correspondence should be unique. E.g. $p_2 p_7 p_{11} \in [L, R] \iff 3$ factors $\iff y = 3$. For the degenerate case of $y = 0$, $\mathsf{fprod} = 1$.

The following lemma shows that such Hprime, L, R exist and specifies their parameters:

Lemma 1. *Assume a collsion-resistant function that maps integers to prime numbers,* $\mathsf{Hprime} : [n] \to \left(2^{\kappa(\lambda)}, 2^{\kappa(\lambda)+\frac{\kappa(\lambda)}{n}}\right)$*, parametrized by* λ *and* n*, and functions* $L : \{0, \ldots, n\} \to \mathbb{Z}$*,* $R : \{0, \ldots, n\} \to \mathbb{Z}$ *such that* $L(y) = 2^{\kappa y}$ *and* $R(y) = 2^{(\kappa+\frac{\kappa}{n})y}$ *respectively. Then for any* $\mathcal{I} \subseteq [n]$*:*

$$\prod_{i \in \mathcal{I}} \mathsf{Hprime}(i) \in [L(y), R(y)] \iff |\mathcal{I}| = y$$

Proof. For any number of factors $y = 1, \ldots, n$ we have $2^{\kappa y} < \prod_{i \in [y]} p_i < 2^{(\kappa+\frac{\kappa}{n})y}$. Since $\kappa y + \frac{\kappa y}{n} < \kappa(y+1)$ for any $y \in [n]$ all ranges are distinct. So the mapping is '1-1'.

In Section 3.3 we discuss concrete instantiations for the function Hprime and consequently L and R.

3.2 Security

Correctness. Follows from correctness of the [CFG$^+$20] Vector Commitment, correctness of $\mathsf{PoKE}, \mathsf{PoDDH}, \mathsf{PoRE}$ arguments of knowledge and from Lemma 1.

Function Binding. Our proof strategy is the following. Given two openings Λ and Λ' of the same commitment C to distinct outputs y, y', we first use the 'subvector' proofs' extractors $\pi_0, \pi_1, \pi'_0, \pi'_1$ to argue that (the exponents of) (F_0, F_1) and (F'_0, F'_1) are subvectors of C. Then we use the PoDDH's extractors π_2, π'_2 to argue that in fact these subvectors are for the same subset of positions $\mathcal{I}_1 = \{i \in [n] : f_i = 1\}$. For the latter we also use the collision-resistance of Hprime. Then we use the extractors of PoRE π_3, π'_3 for F_1 and F'_1 resp., the fact that $y \neq y'$ (by definition of the game) and lemma 1 to argue that these subvectors (for the same positions) are different. Finally, we argue that this fact, having two different subvectors for the same subset of positions and commitment C, contradicts the position-binding property of the [CFG$^+$20] Vector Commitment.

Theorem 1. *Let* Ggen *be a hidden order group generator where the [CFG$^+$20] VC is position binding,* $\mathsf{PoKE}, \mathsf{PoDDH}, \mathsf{PoRE}$ *be succinct knowledge-extractable arguments of knowledge and* Hprime *be collision-resistant. Then our functional commitment for binary inner products is function binding.*

Proof. We organize the proof in hybrid arguments. To start with, we define the game G_0 as the original functional binding game of Definition 1, and our goal is to prove that for any PPT adversary $\mathcal{A}$ and any $n \in \mathbb{N}$, $\Pr[G_0 = 1] \in \mathsf{negl}(\lambda)$.

Game G_0:

$G_0 = \mathsf{FuncBind}_{\mathcal{A},\mathsf{FC}}(\lambda)$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$\left(C, \boldsymbol{f}, y, \Lambda, y', \Lambda'\right) \leftarrow \mathcal{A}(\mathsf{pp})$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda', \boldsymbol{f}, y')$

return b

For any adversary $\mathcal{A}$ against G_0, there exist the extractors of the proof of 'subvector' $\pi_0, \pi_1, \pi'_0, \pi'_1$.

Game G_1: is the same as G_0 except that we execute the extractors $\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}'_0, \mathcal{E}'_1$ of $\pi_0, \pi_1, \pi'_0, \pi'_1$, which output (W_0, fprod_0), (W_1, fprod_1), $\left(W'_0, \mathsf{fprod}'_0\right)$ and $\left(W'_1, \mathsf{fprod}'_1\right)$, respectively.

G_1

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$(C, \boldsymbol{f}, y, \Lambda, y', \Lambda') \leftarrow \mathcal{A}(\mathsf{pp})$

$\boxed{\mathsf{fprod}_0 \leftarrow \mathcal{E}_0(\mathbb{G}, g_0); \mathsf{fprod}_1 \leftarrow \mathcal{E}_1(\mathbb{G}, g_1)}$

$\boxed{\mathsf{fprod}'_0 \leftarrow \mathcal{E}'_0(\mathbb{G}, g_0); \mathsf{fprod}'_1 \leftarrow \mathcal{E}'_1(\mathbb{G}, g_1)}$

$$\boxed{b_{\mathsf{wit}} = \bigwedge_{i=0,1} \left((W_i^{\mathsf{fprod}_i} = C_i) \wedge (W'^{\mathsf{fprod}'_i}_i = C_i) \wedge (g_i^{\mathsf{fprod}_i} = F_i) \wedge (g_i^{\mathsf{fprod}'_i} = F'_i) \right)}$$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda', \boldsymbol{f}, y')$

$\boxed{\textbf{if } b_{\mathsf{wit}} = 0 \textbf{ then } b \leftarrow 0}$

return b

where above $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$, $\Lambda' = (F'_0, F'_1, W'_0, W'_1, \pi'_0, \pi'_1, \pi'_2, \pi'_3)$ and $\mathsf{pp}_n = (\mathbb{G}, g, g_0, g_1, \mathsf{Hprime}, U_n)$.

It is easy to see that the games G_0 and G_1 are identical except if $b_{\mathsf{wit}} = 0$. However, $b_{\mathsf{wit}} = 0$ only when one of the witnesses returned by the extractors is incorrect. By the knowledge extractability of the proof of 'subvector' we obtain that

$$\Pr[G_0 = 1] - \Pr[G_1 = 1] \leq \Pr[b_{\mathsf{wit}} = 0] \in \mathsf{negl}(\lambda).$$

Game G_2: is the same as G_1 except the case $\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 \neq \mathsf{fprod}$ or $\mathsf{fprod}'_0 \cdot \mathsf{fprod}'_1 \neq \mathsf{fprod}$.

G_2

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$(C, \boldsymbol{f}, y, \Lambda, y', \Lambda',) \leftarrow \mathcal{A}(\mathsf{pp})$

$\mathsf{fprod}_0 \leftarrow \mathcal{E}_0(\mathbb{G}, g_0); \mathsf{fprod}_1 \leftarrow \mathcal{E}_1(\mathbb{G}, g_1)$

$\mathsf{fprod}'_0 \leftarrow \mathcal{E}'_0(\mathbb{G}, g_0); \mathsf{fprod}'_1 \leftarrow \mathcal{E}'_1(\mathbb{G}, g_1)$

$$b_{\mathsf{wit}} = \bigwedge_{i=0,1} \left((W_i^{\mathsf{fprod}_i} = C_i) \wedge (W'^{\mathsf{fprod}'_i}_i = C_i) \wedge (g_i^{\mathsf{fprod}_i} = F_i) \wedge (g_i^{\mathsf{fprod}'_i} = F'_i) \right)$$

$$\boxed{b_{\mathsf{col}} = (\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 = \mathsf{fprod}) \wedge (\mathsf{fprod}'_0 \cdot \mathsf{fprod}'_1 = \mathsf{fprod})}$$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}_n, C, \Lambda', \boldsymbol{f}, y')$

$\boxed{\textbf{if } b_{\mathsf{wit}} = 0 \vee b_{\mathsf{col}} = 0 \textbf{ then } b \leftarrow 0}$

return b

where above $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$, $\Lambda' = (F'_0, F'_1, W'_0, W'_1, \pi'_0, \pi'_1, \pi'_2, \pi'_3)$ and $\mathsf{pp}_n = (\mathbb{G}, g, g_0, g_1, \mathsf{Hprime}, U_n)$.

Lemma 2. *Assume that the Low order assumption holds for* Ggen*, then* $\Pr[G_1 = 1] - \Pr[G_2 = 1] \in \mathsf{negl}(\lambda)$.

Proof. We consider a game G_2' by running the extractors of the PoDDH proofs π_2 and π_2', which outputs $(\overline{\mathsf{fprod}_0}, \overline{\mathsf{fprod}_1})$ s.t. $g_0^{\overline{\mathsf{fprod}_0}} = F_0 \wedge g_1^{\overline{\mathsf{fprod}_1}} = F_1 \wedge g^{\overline{\mathsf{fprod}_0}\cdot\overline{\mathsf{fprod}_1}} = F$ and $(\overline{\mathsf{fprod}_0}', \overline{\mathsf{fprod}_1}')$ s.t. $g_0^{\overline{\mathsf{fprod}_0}'} = F_0' \wedge g_1^{\overline{\mathsf{fprod}_1}'} = F_1' \wedge g^{\overline{\mathsf{fprod}_0}'\cdot\overline{\mathsf{fprod}_1}'} = F$, respectively. It is easy to see that $\Pr[G_2 = 1] - \Pr[G_2' = 1] \in \mathsf{negl}(\lambda)$. The following equalities hold:

$$g^{\overline{\mathsf{fprod}_0}\cdot\overline{\mathsf{fprod}_1}} = g^{\overline{\mathsf{fprod}_0}'\cdot\overline{\mathsf{fprod}_1}'} = F;$$

$$g_0{}^{\overline{\mathsf{fprod}_0}} = F_0; \quad g_0{}^{\overline{\mathsf{fprod}_0}'} = F_0';$$

$$g_1{}^{\overline{\mathsf{fprod}_1}} = F_1 \text{ and } g_1{}^{\overline{\mathsf{fprod}_1}'} = F_1'.$$

The game G_2' only differs from G_1 in the case $(\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 \neq \mathsf{fprod}) \vee \big(\mathsf{fprod}_0' \cdot \mathsf{fprod}_1' \neq \mathsf{fprod}\big)$. We divide it into two cases.

Case 1: $\overline{\mathsf{fprod}_0} \cdot \overline{\mathsf{fprod}_1} \neq \mathsf{fprod}$ or $\overline{\mathsf{fprod}_0'} \cdot \overline{\mathsf{fprod}_1'} \neq \mathsf{fprod}$. The first inequality implies that there exists $1 \neq v = \mathsf{fprod} - \overline{\mathsf{fprod}_0} \cdot \overline{\mathsf{fprod}_1} < 2^{\mathsf{poly}(\lambda)}$ such that $g^v = 1$, which is a violation to the Low order assumption and occurs with a negligible probability. The second one implies a similar result.

Case 2: the two above equalities hold. Then G_2' differs from G_1 only if one of the four following inequalities holds.

$$\overline{\mathsf{fprod}_0} \neq \mathsf{fprod}_0; \overline{\mathsf{fprod}_0}' \neq \mathsf{fprod}_0'; \overline{\mathsf{fprod}_1} \neq \mathsf{fprod}_1 \text{ or } \overline{\mathsf{fprod}_1}' \neq \mathsf{fprod}_1'.$$

Again, any of these inequalities imply a low order root of 1.

Therefore, $\Pr[G_1 = 1] - \Pr[G_2 = 1] \in \mathsf{negl}(\lambda)$.

Game G_3: is obtained by adding the range check for fprod_1 and fprod_1' to G_2.

G_3

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$(C, \boldsymbol{f}, y, \Lambda, y', \Lambda',) \leftarrow \mathcal{A}(\mathsf{pp})$

$\mathsf{fprod}_0 \leftarrow \mathcal{E}_0(\mathbb{G}, g_0); \mathsf{fprod}_1 \leftarrow \mathcal{E}_1(\mathbb{G}, g_1)$

$\mathsf{fprod}_0' \leftarrow \mathcal{E}_0'(\mathbb{G}, g_0); \mathsf{fprod}_1' \leftarrow \mathcal{E}_1'(\mathbb{G}, g_1)$

$b_{\mathsf{wit}} = \bigwedge_{i=0,1} \Big((W_i^{\mathsf{fprod}_i} = C_i) \wedge (W'^{\mathsf{fprod}_i'}_i = C_i) \wedge (g_i^{\mathsf{fprod}_i} = F_i) \wedge (g_i{}^{\mathsf{fprod}_i'} = F_i') \Big)$

$b_{\mathsf{col}} = (\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 = \mathsf{fprod}) \wedge \big(\mathsf{fprod}_0' \cdot \mathsf{fprod}_1' = \mathsf{fprod}\big)$

$\boxed{b_{\mathsf{range}} = (\mathsf{fprod}_1 \in [L(y), R(y)]) \wedge \big(\mathsf{fprod}_1' \in [L(y'), R(y')]\big)}$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}, C, \Lambda', \boldsymbol{f}, y')$

$\boxed{\textbf{if } b_{\mathsf{wit}} = 0 \vee b_{\mathsf{col}} = 0 \vee b_{\mathsf{range}} = 0 \textbf{ then } b \leftarrow 0}$

return b

where above $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$, $\Lambda' = (F_0', F_1', W_0', W_1', \pi_0', \pi_1', \pi_2', \pi_3')$ and $\mathsf{pp}_n = (\mathbb{G}, g, g_0, g_1, \mathsf{Hprime}, U_n)$.

The two games G_2 and G_3 are different only in the case the range check fails, which means either π_3 or π_3' is incorrect. By the argument of knowledge of PoRE, we conclude that $\Pr[G_2 = 1] - \Pr[G_3 = 1] \in \mathsf{negl}(\lambda)$.

Game G_4: is the same as G_3 except we decode the subvectors $\boldsymbol{s}, \boldsymbol{s'}$ from the exponents fprod_1 and fprod'_1 corresponding to the subset $\{i : f_i = 1\}$. Then we recompute the exponents from the prime products at 0-positions and 1-positions and check if they coincide with fprod_0, fprod_1, fprod'_0 and fprod'_1.

G_4

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{pp}_n \leftarrow \mathsf{Specialize}(\mathsf{pp}, n)$

$(C, \boldsymbol{f}, y, \Lambda, y', \Lambda') \leftarrow \mathcal{A}(\mathsf{pp})$

$\mathsf{fprod}_0 \leftarrow \mathcal{E}_0(\mathbb{G}, g_0); \mathsf{fprod}_1 \leftarrow \mathcal{E}_1(\mathbb{G}, g_1)$

for $i : f_i = 1$ **do**

$p_i \leftarrow \mathsf{Hprime}(i); s_i = (p_i \mid \mathsf{fprod}_1); s'_i = (p_i \mid \mathsf{fprod}'_1)$

$\boldsymbol{s} = \{s_i\}_{f_i=1}; \boldsymbol{s'} = \{s'_i\}_{f_i=1}$

$\mathsf{fprod}'_0 \leftarrow \mathcal{E}'_0(\mathbb{G}, g_0); \mathsf{fprod}'_1 \leftarrow \mathcal{E}'_1(\mathbb{G}, g_1)$

$b_{\mathsf{wit}} = \bigwedge_{i=0,1} \left((W_i^{\mathsf{fprod}_i} = C_i) \wedge (W'^{\mathsf{fprod}'_i}_i = C_i) \wedge (g_i^{\mathsf{fprod}_i} = F_i) \wedge (g_i^{\mathsf{fprod}'_i} = F'_i) \right)$

$b_{\mathsf{col}} = (\mathsf{fprod}_0 \cdot \mathsf{fprod}_1 = \mathsf{fprod}) \wedge \left(\mathsf{fprod}'_0 \cdot \mathsf{fprod}'_1 = \mathsf{fprod}\right)$

$b_{\mathsf{range}} = (\mathsf{fprod}_1 \in [L(y), R(y)]) \wedge \left(\mathsf{fprod}'_1 \in [L(y'), R(y')]\right)$

$$b_{\mathsf{subv}} = \left(\mathsf{fprod}_0 = \prod_{f_i=1, s_i=0} \mathsf{Hprime}(i) \wedge \mathsf{fprod}'_0 = \prod_{f_i=1 : s'_i=0} \mathsf{Hprime}(i) \right)$$

$b \leftarrow \mathsf{Ver}(\mathsf{pp}, C, \Lambda, \boldsymbol{f}, y) = 1 \wedge y \neq y' \wedge \mathsf{Ver}(\mathsf{pp}, C, \Lambda', \boldsymbol{f}, y')$

if $b_{\mathsf{subv}} = 0 \vee b_{\mathsf{wit}} = 0 \vee b_{\mathsf{col}} = 0 \vee b_{\mathsf{range}} = 0$ **then** $b \leftarrow 0$

return b

where above $\Lambda = (F_0, F_1, W_0, W_1, \pi_0, \pi_1, \pi_2, \pi_3)$, $\Lambda' = (F'_0, F'_1, W'_0, W'_1, \pi'_0, \pi'_1, \pi'_2, \pi'_3)$, $\mathsf{pp}_n = (\mathbb{G}, g, g_0, g_1, \mathsf{Hprime}, U_n)$.

Lemma 3. *Assume that* Hprime *is collision-resistant,* $\Pr[G_3 = 1] - \Pr[G_4 = 1] \in \mathsf{negl}(\lambda)$.

Proof. The output of G_4 differs from G_3's only when $b_{\mathsf{subv}} = 0 \wedge b_{\mathsf{wit}} = 1 \wedge b_{\mathsf{col}} = 1$. By assumption, for all i such that $f_i = 1$, $\mathsf{Hprime}(i)$'s are all distinct. Then by definition of $\boldsymbol{s}$,

$$\prod_{f_i=1, s_i=1} \mathsf{Hprime}(i) \mid \mathsf{fprod}_1 \text{ and } \prod_{f_i=1, s_i=0} \mathsf{Hprime}(i) \mid \mathsf{fprod}_0. \tag{1}$$

Multiplying both hand sides of the two equations, we obtain $\prod_{f_i=1} \mathsf{Hprime}(i) \mid \mathsf{fprod}$. Notice that the equality holds, hence it also holds for the equations in (1).

Lemma 4. *Let* Ggen *be a hidden order group generator where the [CFG$^+$20] SVC is position binding, then* $\Pr[G_4 = 1] \in \mathsf{negl}(\lambda)$.

Proof. We show that given any adversary $(\mathcal{A}, \mathcal{E})$, where $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}'_0, \mathcal{E}'_1)$, winning game G_4 with non-negligible advantage, we can construct $\mathcal{B}$ winning the position-binding game of the [CFG$^+$20] VC.

Indeed, $\mathcal{B}$ on input pp outputs $(C, \mathcal{I}_1, \boldsymbol{s}, (W_0, W_1), \boldsymbol{s'}, (W'_0, W'_1))$, where $\mathcal{I}_1 = \{i \in [n] : f_i = 1\}$ and $(W_0, W_1), (W'_0, W'_1)$ play the role of the (accepting) opening proofs for the [CFG$^+$20] SVC. The verifier for position binding computes $a_j = \prod_{f_i=1, s_i=j} \mathsf{Hprime}(i); a'_j = \prod_{f_i=1, s'_i=j} \mathsf{Hprime}(i)$ for $j = 0, 1$. Since $b_{\mathsf{subv}} = 1$, $a_j = \mathsf{fprod}_j$ and $a'_j = \mathsf{fprod}'_j$ for $j = 0, 1$. Next, since $b_{\mathsf{wit}} = 1$, $W_j^{a_j} = C_j$ and $W_j'^{a'_j} = C_j$, for $j = 0, 1$.

Now since $y \neq y'$, $b_{\mathsf{range}} = 1$ and from lemma 1 we infer that $\mathsf{fprod}_1 \neq \mathsf{fprod}'_1$ (because they have different number of factors) so $\boldsymbol{s} \neq \boldsymbol{s'}$. Combining these arguments, we obtain that $\mathsf{Ver}(C, \mathcal{I}_1, \boldsymbol{s}, , (W_0, W_1)) = 1$, $\mathsf{Ver}(C, \mathcal{I}_1, \boldsymbol{s'}, , (W'_0, W'_1)) = 1$ (for the VC) and $\boldsymbol{s} \neq \boldsymbol{s'}$, i.e., $\mathcal{B}$ succeeds in the position-binding game with the same probability of $(\mathcal{A}, \mathcal{E})$.

By combining all the lemmas we conclude that any PPT adversary has at most negligible probability of breaking the function binding of our SVC scheme.

3.3 Instantiation

Instatiation of Hprime**.** As stated in Lemma 1, Hprime should be a collision-resistant function with domain $[n]$ that outputs prime numbers in the range $\left(2^{\kappa(\lambda)}, 2^{\kappa(\lambda)+\frac{\kappa(\lambda)}{n}}\right)$. Here we specify the function $\kappa(\cdot)$ and show instantiations for Hprime under these restrictions.

Hashing to primes is a well studied problem [GHR99, CS99, CMS99]. A standard technique is rejection sampling: on input x it computes $y = F_K(x, 0)$, where F_K is a pseudorandom function with seed K and range $[A, B]$, and checks y for primality. If y is not prime it continues to $y = F_K(x, 1)$ and so on, until it finds a prime $F(x, j)$ for some j. From the density of primes the expected number of tries is $\log(B - A)$. As an alternative, F_K can also be a random oracle.

Assume a collision-resistant hash function H that we model as a random oracle and its outputs are in the range $\left(2^{\kappa(\lambda)}, 2^{\kappa(\lambda)+\frac{\kappa(\lambda)}{n}}\right)$.[9] For H to be collision resistant we require, due to the birthday bound, its range to contain at least $2^{2\lambda}$ prime numbers. From the density of primes we know that in the above range there are about:

$$\frac{2^{\kappa+\frac{\kappa}{n}}}{\kappa+\frac{\kappa}{n}} - \frac{2^{\kappa}}{\kappa} \approx \frac{2^{\kappa+\frac{\kappa}{n}}}{\kappa} - \frac{2^{\kappa}}{\kappa} = \frac{2^{\kappa}\left(2^{\frac{\kappa}{n}} - 1\right)}{\kappa}$$

prime numbers, where for the first approximate equality we assumed that $\kappa \ll n$.

[9] We can securely fix the range of a hash function (as SHA512) by fixing some of its bits and truncating others.

So if we set κ (depending on λ and n) to be the smallest positive integer such that:

$$\frac{2^{\kappa}\left(2^{\frac{\kappa}{n}}-1\right)}{\kappa} \geq 2^{2\lambda}$$

then H gives sufficiently many primes to instantiate Hprime (via the rejection sampling method we described above).

For example for $n = 2^{60}$ and $\lambda = 128$: $\kappa = 317$. So instantiating Hprime with the rejection sampling method based on the SHA512 (for F_K) fixing its output range to $(2^{317}, 2^{317+317/2^{60}})$ we get a sufficient Hprime for our functional vector commitment construction that satisfies lemma 1.

Instantiation of the Arguments of Knowledge. Here we present the merged argument of knowledge for our Open algorithm of section 3.1. As noted, it was presented modularly in order to ease the presentation of the protocol and its security proof, however we can merge the proofs for the four languages $\mathcal{L}_0 - -\mathcal{L}_3$ into a single protocol, using standard composition techniques. The unified language of the Open algorithm is:

$$\mathcal{L} = \left\{ \begin{array}{l} (F_0, F_1, F, W_0, W_1, R, L; \mathsf{fprod}_0, \mathsf{fprod}_1) : \\ \qquad\qquad W_0^{\mathsf{fprod}_0} = C_0 \wedge g_0^{\mathsf{fprod}_0} = F_0 \wedge \\ \qquad\qquad \wedge W_1^{\mathsf{fprod}_1} = C_1 \wedge g_1^{\mathsf{fprod}_1} = F_1 \wedge \\ \qquad\qquad \wedge g^{\mathsf{fprod}_0 \cdot \mathsf{fprod}_1} = F \wedge L < \mathsf{fprod}_1 < R \end{array} \right\}$$

The description of the protocol is in Figure 3.

The protocol gets non-interactive after the Fiat-Shamir transform. We note that for ℓ an instantiation of the random oracle with a hash-to-prime function of outputs in $\mathsf{Primes}(2\lambda)$ suffices.

Concrete Security Assumptions After the above instantiations we get that our overall binary inner product commitment is secure in the GGM and RO model assuming that H (used for Hprime as described above) is collision-resistant: the argument of knowledge of Figure 3 is knowledge-extractable in the generic group model and gets non-interactive in the random oracle model and the [CFG+20] SVC is position binding under the 2-strong RSA and Low order assumptions (that are secure in the GGM).

3.4 Efficiency

Our FC for binary inner products has $O(1)$ public parameters, $O(1)$ commitment size and $O(1)$ openings proof size. More in detail, $|\mathsf{pp}_n|$ consists of 4 $|\mathbb{G}|$-elements and the descriptions of $\mathbb{G}$ and Hprime (which are concise). $|C|$ is 5 $|\mathbb{G}|$-elements and 2 $|\mathbb{Z}_{2^{2\lambda}}|$-elements. Finally the opening proof $|\Lambda|$ is 21 $|\mathbb{G}|$-elements and 7 $|\mathbb{Z}_{2^{2\lambda}}|$-elements.[10]

[10] We do not consider optimizations for the arguments of knowledge with which we could reduce the size of $|C|$ by 1 and $|\Lambda|$ by 6 group elements respectively.

$\mathsf{Setup}(\lambda)$: $\mathbb{G} \leftarrow\!\$\ \mathsf{Ggen}(\lambda)$, $g, g_0, g_1 \leftarrow\!\$\ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g, g_0, g_1)$.

$\underline{\mathsf{P} \to \mathsf{V}}$: Computes:
- $F_1' = g_1^{(\mathsf{fprod}_1 - L)(R - \mathsf{fprod}_1)}$
- For $i = 1, 2, 3$: $Z_{x_i} = g_1^{x_i^2}$, $z_{x_i} = g_1^{x_i}$

Sends $(F_1', \{Z_{x_i}, z_{x_i}\}_{i=1}^3)$
$\underline{\mathsf{V} \to \mathsf{P}}$: Sends $\ell \leftarrow\!\$\ \mathsf{Primes}(\lambda)$
$\underline{\mathsf{P} \to \mathsf{V}}$: Computes:
- For $b = 0, 1$: $q_b = \lfloor \mathsf{fprod}_b / \ell \rfloor$, $Q_b = W_b^{q_b}$, $Q_b' = g_b^{q_b}$, $r_b = \mathsf{fprod}_b \mod \ell$.
- $q = \lfloor \mathsf{fprod}_0 \cdot \mathsf{fprod}_1 / \ell \rfloor$, $Q = g^q$, $r = \mathsf{fprod}_0 \cdot \mathsf{fprod}_1 \mod \ell$
- For $i = 1, 2, 3$: $q_{x_i} = \lfloor x_i / \ell \rfloor$, $Q_{x_i} = z_i^{q_{x_i}}$, $Q_{x_i}' = g_i^{q_{x_i}}$, $r_{x_i} \mod \ell$
- $q_L = \lfloor (\mathsf{fprod}_1 - L)/\ell \rfloor$, $Q_L = g_1^{q_L}$, $r_L = (\mathsf{fprod}_1 - L) \mod \ell$
- $q_R = \lfloor (R - \mathsf{fprod}_1)/\ell \rfloor$, $Q_R = g_1^{q_R}$, $r_R = (R - \mathsf{fprod}_1) \mod \ell$
- $q_{LR} = \lfloor (\mathsf{fprod}_1 - L) \cdot (R - \mathsf{fprod}_1)/\ell \rfloor$, $Q_{LR} = g_1^{q_{LR}}$, $r_{LR} = (\mathsf{fprod}_1 - L) \cdot (R - \mathsf{fprod}_1) \mod \ell$

Sends $\pi := \big(\{Q_b, Q_b', r_b\}_{b=0}^1, Q, \{Q_{x_i}, Q_{x_i}', r_{x_i}\}_{i=1}^3, Q_L, r_L, Q_R, r_R, Q_{LR}\big)$.
$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Outputs 1 iff:
- $r_0, r_1, r_{x_1}, r_{x_2}, r_{x_3}, r_L, r_R \in [\ell]$
- $Q_b'^{\ell} g_b^{r_b} = F_b$, $Q_b^{\ell} W_b^{r_b} = C_b$, for $b = 0, 1$
- $Q^{\ell} g^r = F$
- $Q_{x_i}'^{\ell} g_1^{r_{x_i}} = z_i$, $Q_{x_i}^{\ell} z_{x_i}^{r_{x_i}} = Z_{x_i}$, for $i = 1, 2, 3$
- $Q_L^{\ell} g_1^{r_L} = F_1 \cdot g_1^{-L}$, $Q_R^{\ell} g_1^{r_R} = g_1^R \cdot F_1^{-1}$, $Q_{LR}^{\ell} g_1^{r_{LR}} = F_1'$
- $F_1'^4 \cdot g_1 = \prod_{i=1}^3 Z_i$

where $r = r_0 r_1 \mod \ell$ and $r_{LR} = r_L r_R \mod \ell$.

Fig. 3: Our merged protocol for the Open algorithm of our binary inner product Functional Commitment (section 3.1). The proof size and verification time are independent of the size of fprod and thus n.

Generating the public parameters, via Setup and $\mathsf{Specialize}$, takes a $\mathbb{G}$-exponentiation of size $\kappa n = O_\lambda(n)$. The generation doesn't require any private coins and thus is transparent.

The prover's time (i.e. the running time of Open) is dominated by the exponentation $F_1' = g_1^{(\mathsf{fprod}_1 - L)(R - \mathsf{fprod}_1)}$, that is a $\mathbb{G}$-exponentation of size $2\kappa n = O_\lambda(n)$.

The verifier's running time (i.e. Ver) is dominated by the computation of $F = g^{\prod_{f_i=1} \mathsf{Hprime}(i)}$, which takes (in the worst case where $\boldsymbol{f} = (1, 1, \ldots, 1)$) a $\mathbb{G}$-exponentiation of size $\kappa n = O_\lambda(n)$. The rest of the computations, i.e. the verification of the argument of knowledge, take constant time $O(2\lambda) = O_\lambda(1)$ $\mathbb{G}$-exponentiations. However, below we make two observations that can speed up the verification in two useful ways.

Preprocessing-based verification. Our Functional Commitment construction allows preprocessing the verification (see Remark 2). The verifier can compute a-priori the function-dependent value F so that the online verification gets $O_\lambda(n)$. Notably, this preprocessing is deterministic and proof-independent, and thus can be reused to verify an unbounded number of openings for the same function f.

From group-based to integers-based linear work. Even without preprocessing, the prover can compute and send to the verifier a PoE proof [Wes19, BBF19] (see Appendix A) for $F = g^{\prod_{f_i=1} \mathsf{Hprime}(i)}$. Then the verifier verifies PoE instead of computing F herself. This takes $O_\lambda(n)$ integer operations and $O_\lambda(1)$ group exponentiations, in place of $O_\lambda(n)$ group operations, which concretely gives a significant saving.

4 Our FC for Inner Products Over the Integers

In this section, we present two transformations that turn any functional commitment for binary inner products (like the one we presented in Section 3) into a functional commitment for inner products of vectors of (bounded) integers. Precisely, we build an FC where one commits to vectors $\boldsymbol{v} \in (\mathbb{Z}_{2^\ell})^n$ and the class of admissible functions is

$$\mathcal{F}_n = \{f : (\mathbb{Z}_{2^\ell})^n \to \mathbb{Z}\}$$

where each f is represented as a vector $\boldsymbol{f} \in (\mathbb{Z}_{2^m})^n$.

Consider an FC scheme bitFC for binary inner products, and let $t_{\mathsf{Com}}(n)$, $t_{\mathsf{Open}}(n)$, $t_{\mathsf{Ver}}(n)$ be the running times of its algorithms Com, Open and Ver respectively, and let $s(n)$ be the size of its openings.

Our two transformations yield FCs for the integer inner products functionality $\mathcal{F}$ that achieve different tradeoffs:

1. With our first transformation we obtain an FC where

$$t'_{\mathsf{Com}}(n) = t_{\mathsf{Com}}(n\ell),\ t'_{\mathsf{Open}}(n) = (\ell+m)\cdot t_{\mathsf{Com}}(n\ell),\ t'_{\mathsf{Ver}}(n) = (\ell+m)\cdot t_{\mathsf{Ver}}(n\ell),$$

$$s'(n) = (\ell+m)\cdot(s(n\ell)+\log(n\ell))$$

2. With our second transformation we obtain an FC where

$$t'_{\mathsf{Com}}(n) = t_{\mathsf{Com}}(n\ell 2^{\ell+m}),\ t'_{\mathsf{Open}}(n) = t_{\mathsf{Com}}(n\ell 2^{\ell+m}),\ t'_{\mathsf{Ver}}(n) = t_{\mathsf{Ver}}(n\ell 2^{\ell+m}),$$

$$s'(n) = s(n\ell 2^{\ell+m})$$

Given the tradeoffs, and considering instantiations of bitFC like ours, in which $s(n)$ is a fixed value in the security parameters, then the second transformation is particularly interesting in the case $\ell, m = O(1)$ are constant or $O(\log\lambda)$ as it yields an FC with constant, or polynomial, time overhead and constant-size openings.

4.1 Our lifting to FC for integer inner products with logarithmic-size openings

We start by providing here an intuitive description of our transformation. We give a formal description of the FC scheme slightly below.

For our transformation, we use a binary representation of the vectors of integers $\boldsymbol{v} \in (\mathbb{Z}_{2^\ell})^n$ and $\boldsymbol{f} \in (\mathbb{Z}_{2^m})^n$, that is:

$$\boldsymbol{v} = (v_1, \ldots, v_n) \in (\{0,1\}^\ell)^n \quad \text{and} \quad \boldsymbol{f} = (f_1, \ldots, f_n) \in (\{0,1\}^m)^n$$

Denote $v_i = \sum_{j=0}^{\ell-1} v_i^{(j)} 2^j$ and $f_i = \sum_{k=0}^{m-1} f_i^{(k)} 2^k$ the bit decomposition of v_i, f_i respectively. Then we can rewrite the inner product of $\boldsymbol{v}$ and $\boldsymbol{f}$ as

$$y = \langle \boldsymbol{f}, \boldsymbol{v} \rangle = \sum_{i=1}^{n} v_i f_i = \sum_{i=1}^{n} \left(\sum_{j=0}^{\ell-1} \sum_{k=0}^{m-1} v_i^{(j)} f_i^{(k)} 2^{j+k} \right). \tag{2}$$

If we swap the counters we conclude to:

$$y = \sum_{j=0}^{\ell-1} \sum_{k=0}^{m-1} \left(\sum_{i=1}^{n} v_i^{(j)} f_i^{(k)} \right) 2^{j+k} = \sum_{j=0}^{\ell-1} \sum_{k=0}^{m-1} \langle \boldsymbol{v}^{(j)}, \boldsymbol{f}^{(k)} \rangle 2^{j+k}$$

where above $\boldsymbol{v}^{(j)} = (v_1^{(j)}, \ldots, v_n^{(j)}) \in \{0,1\}^n$ is a bit-vector of the j-th bits of all entries v_i (and similarly for $\boldsymbol{f}^{(k)}$). The inner product y of $\boldsymbol{v}$ and $\boldsymbol{f}$ is hereby broken into the above sum of ℓm binary inner products. So, a first idea to open the inner product over the integers would be to let one create ℓ commitments, one for each $\boldsymbol{v}^{(j)}$ of length n, and then open to the inner product y by revealing all the ℓm binary inner products, each with its corresponding opening proof. The issue with this idea is that it yields an $O(\ell m \log n)$-size opening.

Next, we show a more efficient way to use an FC for binary inner product that avoids this quadratic blowup.

To this end, we show that y can also be represented as the sum of $\ell + m$ binary inner products between vectors of length $n\ell$. We start observing that we can rewrite (2) as

$$y = \sum_{i=1}^{n} \sum_{j=0}^{\ell-1} v_i^{(j)} \cdot \left(\sum_{k=0}^{m-1} f_i^{(k)} 2^{j+k} \right) = \sum_{i=1}^{n} \sum_{j=0}^{\ell-1} v_i^{(j)} \cdot \hat{f}_i^{(j)} \tag{3}$$

where, for every i, j, each $\hat{f}_i^{(j)}$ is the integer $\sum_{k=0}^{m-1} f_i^{(k)} \cdot 2^{j+k} \in [0, 2^{\ell+m-1}]$. Now the inner product y over integers is reshaped as an inner product between an $n\ell$-long binary vector

$$\boldsymbol{v}' = \boldsymbol{v}^{(0)} \| \ldots \| \boldsymbol{v}^{(\ell-1)}$$

and an $n\ell$-long function with coefficients in $[0, 2^{\ell+m-1}]$. It is left to appropriately grind this inner product into binary inner products. We are about to show that those binary inner products are between $\boldsymbol{v}'$ and the following binary vectors

$$\boldsymbol{f}'_h = \boldsymbol{f}^{(h)} \| \boldsymbol{f}^{(h-1)} \| \ldots \| \boldsymbol{f}^{(h-\ell+1)}$$

where $h \in [0, \ell+m-2]$, $\boldsymbol{f}^{(k)} = (0, \ldots, 0)$ for all $k \notin [0, m-1]$.

Indeed, let $y'_h = \langle \boldsymbol{v}', \boldsymbol{f}'_h \rangle$. Then by changing the variable $k = h - j$ and rearranging the summation of j, we can rewrite (2) as

$$y = \sum_{h=0}^{\ell+m-2} \sum_{j=0}^{\ell-1} \langle \boldsymbol{v}^{(j)}, \boldsymbol{f}^{(h-j)} \rangle \cdot 2^h = \sum_{h=0}^{\ell+m-2} \langle \boldsymbol{v}', \boldsymbol{f}'_h \rangle \cdot 2^h = \sum_{h=0}^{\ell+m-2} y'_h \cdot 2^h.$$

Using as a building block the binary functional vector commitment we get a functional vector commitment for bounded-integers as follows: only one commitment C is needed for the concatenating vector $\boldsymbol{v}'$. Then the opening proof consists of the partial outputs $\{y_h\}_{h \in [0, \ell+m-2]}$ together with their corresponding functional opening proofs $\{\Lambda_h\}_{h \in [0, \ell+m-2]}$, one for each binary inner product $\langle \boldsymbol{v}', \boldsymbol{f}'_h \rangle$. For verification, one is checking that each Λ_h verifies with respect to C and f'_h, to ensure that y_h are the correct partial outputs. Then it reconstructs $y = \sum_{h=0}^{\ell+m-2} y_h 2^h$ according to the above equality.

FC scheme Consider bitFC as an arbitrary FC for binary inner products, we present below a formal description of the transformation.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: runs $\mathsf{pp} = \mathsf{bitFC.Setup}(1^\lambda)$. Returns pp

$\mathsf{Specialize}(\mathsf{pp}, \mathcal{F}_n) \to \mathsf{pp}_{\mathcal{F}_n}$: given the description of the functions class $\mathcal{F}_n$, which includes the bounds ℓ, m and the vector length n, the specialization algorithms sets $N = n\ell$ and returns $\mathsf{pp}_{\mathcal{F}_n} = \mathsf{bitFC.Specialize}(\mathsf{pp}, N)$.

$\mathsf{Com}(\mathsf{pp}_{n,\ell}, \boldsymbol{v}) \to C$: Let $\boldsymbol{v} = (v_1, \ldots, v_n) \in (\{0,1\}^\ell)^n$ be a vector of ℓ-bit entries, and let $\boldsymbol{v}^{(j)} = (v_1^{(j)}, \ldots, v_n^{(j)})$ be the binary vector expressing the j-th bit of all entries in $\boldsymbol{v}$, i.e., it holds $\boldsymbol{v} = \left(\sum_{j=0}^{\ell-1} v_1^{(j)} 2^j, \ldots, \sum_{j=0}^{\ell-1} v_n^{(j)} 2^j\right)$.

The commitment algorithm computes the commitment

$$C = \mathsf{bitFC.Com}(\mathsf{pp}_{\mathcal{F}_n}, \boldsymbol{v}'), \text{ s.t. } \boldsymbol{v}' = \boldsymbol{v}^{(0)} \| \ldots \| \boldsymbol{v}^{(\ell-1)}$$

and returns C.

$\mathsf{Open}(\mathsf{pp}_{\mathcal{F}_n}, C, \boldsymbol{v}, \boldsymbol{f}) \to (y, \Lambda)$: $\boldsymbol{f} = (f_1, \ldots, f_n) \in (\{0,1\}^m)^n$ is a vector of m-bit entries.

If $\boldsymbol{f} = \left(\sum_{k=0}^{m-1} f_1^{(k)} 2^k, \ldots, \sum_{k=0}^{m-1} f_n^{(k)} 2^k\right)$ then $\boldsymbol{f}^{(k)} = (f_1^{(k)}, \ldots, f_n^{(k)})$ is the binary vector of the k-th bit of all entries of $\boldsymbol{f}$.

The opening algorithm proceeds as follows. For each $h = 0, \ldots, \ell+m-2$:

$$\text{set } \boldsymbol{f}'_h = \boldsymbol{f}^{(h)} \| \boldsymbol{f}^{(h-1)} \| \ldots \| \boldsymbol{f}^{(h-\ell+1)}, \text{ where } \boldsymbol{f}^{(i)} = (0, \ldots, 0) \forall i \notin [0, m-1],$$

$$\text{and compute } y_h = \langle \boldsymbol{v}', \boldsymbol{f}'^h \rangle \text{ and } \Lambda_h = \mathsf{bitFC.Open}(\mathsf{pp}_{\mathcal{F}_n}, C, \boldsymbol{v}', \boldsymbol{f}'_h),$$

Return $\Lambda = \{y_h, \Lambda_h\}_{h \in [0, \ell+m-2]}$.

$\mathsf{Ver}(\mathsf{pp}_{\mathcal{F}_n}, C, \Lambda, \boldsymbol{f}, y) \to b$: returns 1 iff:

1. $\mathsf{bitFC.Ver}\left(\mathsf{pp}_{\mathcal{F}_n}, C, \Lambda_h, \boldsymbol{f}'_h, y_h\right) = 1$, for each $h \in [0, \ell+m-2]$.

2. $y = \sum_{h=0}^{\ell+m-2} y_h 2^h$.

Theorem 2. *If the binary functional vector commitment is functional binding, then our bounded-integer functional vector commitment is functional binding.*

Proof. The proof is straightforward from the fact that two valid openings y, z over integer imply immediately that there exists at least an index h for which there are two valid openings for distinct binary inner products $y_h \neq z_h$.

4.2 Our lifting to FC for integer inner products with constant-size openings

Here we provide a different method to lift an FC for binary inner products to an FC for integer inner products that achieves a different tradeoff. The prover time and verification time are $2^{\ell+m}$ times those of the bitFC scheme, while openings are exactly the same as those of bitFC (and thus constant-size using our scheme of Section 3).

Intuition In the transformation of the previous section we showed how how to express the inner product $y = \langle \boldsymbol{v}, \boldsymbol{f} \rangle$ of n-long vectors of integers into the weighted sum of $\ell + m - 1$ binary inner products of vectors of length $n\ell$:

$$y = \sum_{h=0}^{\ell+m-2} \langle \boldsymbol{v}', \boldsymbol{f}'_h \rangle \cdot 2^h = \sum_{h=0}^{\ell+m-2} y_h \cdot 2^h$$

The drawback of this transformation is that we need to include all the y_h in the opening, and each of this integer is up to $\log n$-bits long.

It turns out that we can iterate the same idea and encode the above weighted sum into a single inner product $\langle \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{f}} \rangle$ of binary vectors of length $n\ell H$ with $H = \sum_{h=0}^{\ell+m-2} 2^h = 2^{\ell+m-1} - 1$.

For every $h \in [0, \ell+m-2]$, define the vector $\tilde{\boldsymbol{f}}_h = \boldsymbol{f}'_h \| \cdot \| \boldsymbol{f}'_h \in \{0,1\}^{n\ell 2^h}$, that is the concatenation of 2^h copies of $\boldsymbol{f}'_h$. Similarly, set $\tilde{\boldsymbol{v}}_h = \boldsymbol{v}' \| \cdots \| \boldsymbol{v}' \in \{0,1\}^{n\ell 2^h}$. Next, if we define

$$\tilde{\boldsymbol{v}} = \tilde{\boldsymbol{v}}_h \| \cdots \| \tilde{\boldsymbol{v}}_h \in \{0,1\}^{n\ell H} \text{ and } \tilde{\boldsymbol{f}} = \tilde{\boldsymbol{f}}_0 \| \cdots \| \tilde{\boldsymbol{f}}_{\ell+m-2} \tag{4}$$

it can be seen that

$$\langle \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{f}} \rangle = \sum_{h=0}^{\ell+m-2} \langle \tilde{\boldsymbol{v}}_h, \tilde{\boldsymbol{f}}_h \rangle = \sum_{h=0}^{\ell+m-2} \langle \boldsymbol{v}', \boldsymbol{f}'_h \rangle \cdot 2^h = y$$

FC Scheme More in detail the FC scheme works as follows.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: runs $\mathsf{pp} = \mathsf{bitFC.Setup}(1^\lambda)$. Returns pp

$\mathsf{Specialize}(\mathsf{pp}, \mathcal{F}_n) \to \mathsf{pp}_{\mathcal{F}_n}$: given the description of the functions class $\mathcal{F}_n$, which includes the bounds ℓ, m and the vector length n, the specialization algorithms sets $N = n\ell H$, with $H = 2^{\ell+m-1}-1$, and returns $\mathsf{pp}_{\mathcal{F}_n} = \mathsf{bitFC.Specialize}(\mathsf{pp}, N)$.

$\mathsf{Com}(\mathsf{pp}_{\mathcal{F}_n}, \boldsymbol{v}) \to C$: Given $\boldsymbol{v} = (v_1, \ldots, v_n) \in (\{0,1\}^\ell)^n$, compute a vector $\tilde{\boldsymbol{v}} \in \{0,1\}^{n\ell H}$ as in equation (4), and return the commitment

$$C = \mathsf{bitFC.Com}(\mathsf{pp}_{\mathcal{F}_n}, \tilde{\boldsymbol{v}}).$$

$\mathsf{Open}(\mathsf{pp}_{\mathcal{F}_n}, C, \boldsymbol{v}, \boldsymbol{f}) \to \Lambda$: Given $\boldsymbol{f} = (f_1, \ldots, f_n) \in (\{0,1\}^m)^n$, compute vectors $\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{f}} \in \{0,1\}^{n\ell H}$ as in equation (4), and return the opening

$$\Lambda = \mathsf{bitFC.Open}(\mathsf{pp}_{\mathcal{F}_n}, \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{f}}).$$

$\mathsf{Ver}(\mathsf{pp}_{\mathcal{F}_n}, C, \Lambda, \boldsymbol{f}, y) \to b$: returns 1 iff $\mathsf{bitFC.Ver}(\mathsf{pp}_{\mathcal{F}_n}, C, \Lambda, \tilde{\boldsymbol{f}}, y) = 1$.

Theorem 3. *If* bitFC *is functional binding, then the FC described above is functional binding.*

The proof is straightforward based on the observation that two valid openings for distinct $y \neq y'$ of our FC are also two valid proofs, for the same commitment and outputs, for the bitFC scheme.

5 Our FC for Inner Products mod p

In this section, we show how to extend the transformations of the previous section in order to build FCs for inner products modulo an integer p, starting from an FC for binary inner products. Namely we build FCs for

$$\mathcal{F}_{p,n} = \{f : (\mathbb{Z}_p)^n \to \mathbb{Z}_p\}.$$

Solutions with logarithmic-size openings. For the FC of our first transformation of Section 4.1, the adaptation to support the inner product mod p is easy. The only change is to run that construction by setting $\ell = m = \|p\|$ and by letting the second verification check be: $y = \sum_{h=0}^{\ell+m-2} y_h \cdot 2^h \bmod p$. Notice that the FC scheme has exactly the same complexity analysis, considering $\ell = m = \|p\|$.

More in general, given any FC for integer inner products it is possible to construct one for inner products modulo an integer p, at the cost of additionally including $\log(np^2)$ bits in the opening: one simply adds to the opening the result y over the integers, and the verifier additionally checks that $y_p = y \bmod p$.

Solutions with constant-size openings. To build an FC for $\mathcal{F}_{p,n}$ in which openings remain of constant size, we discuss two solutions based on our second transformation of Section 4.2.

The first solution is described in Section 5.1. It shows how to use an FC for integer inner products to obtain an FC for inner products modulo p, for $p = \mathsf{poly}(\lambda)$, with no overhead in the size of openings. This construction can be instantiated using the FCs obtained with our second transformation of Section 4.2. To avoid a quadratic blowup in verification time, this construction can start from FC for integer inner products that enjoy preprocessing-based verification.

This way, the verification of the resulting FC remains $O(n)$; as drawback, however the resulting FC does not have preprocessing anymore.

FC *for* $\mathbb{Z}$ *inner products, with* $O(1)$ *proofs and preprocessing*	$\Longrightarrow$	FC *for* $\mathbb{Z}_p$ *inner products, with* $O(1)$ *proofs (no preprocessing)*

The second solution consists into using the same transformation of Section 4.2 with the following differences: set $\ell = m = \|p\|$ and, as a building block, use an FC for binary inner products modulo p, i.e., for computing $\langle \boldsymbol{v}, \boldsymbol{f} \rangle \pmod p$ for $\boldsymbol{v}, \boldsymbol{f} \in \{0,1\}^n$. If such a building block is available and it has constant size proofs, it is easy to see that this variant of the transformation is correct and secure. Clearly, due to the complexity of the transformation we can only use it for small integers $p = O(1), O(\log \lambda)$.

The only missing piece for this construction is showing this building block. In Section 5.2 we describe a construction of such a scheme, obtained by tweaking our scheme of section 3. This solution preserves preprocessing verification.

FC *for* $\mathbb{Z}_2$ *inner products mod* p, *with* $O(1)$ *proofs*	$\Longrightarrow$	FC *for* $\mathbb{Z}_p$ *inner products, with* $O(1)$ *proofs*

5.1 Using FC for integer inner products with preprocessing

As explained above, given an FC for integer inner products intFC (like the ones of Section 4) one can easily build one for inner products $\pmod p$ by including in the opening proof the result y of the inner product over the integers, while the actual result is $y_p = y \mod p$. But y (worst-case) can be $n \cdot p^2$. So the size of the proof gets at least $\|y\| < 2\log(p) + \log(n)$.

To remove the logarithmic dependence on n, we observe that having $y_p = y \mod p$ (which the verifier always has as it is an input of the verification algorithm) suffices for the verifier to check the opening. That is, since the euclidean division of y by p is $y = k \cdot p + y_p$ then $0 \leq k \leq np$. Therefore the verifier, after receiving y_p, can brute-force try all the quotients $k \in [0, np]$, set $y(k) = k \cdot p + y_p$ and check if the proof verifies with respect to the integer $y(k)$. If there is no $k \in [0, np]$ such that the proof verifies, it rejects, otherwise she finds the actual y which is accepting.

Naively, this approach would take time $np \cdot t_{\mathsf{Ver}}(n)$, where $t_{\mathsf{Ver}}(n)$ is the verification time of intFC, when checking inner products of length n. The problem with this is that, for $t_{\mathsf{Ver}}(n) = O_\lambda(n)$, the verification becomes $O_\lambda(n^2)$. However, we notice that the verification time of this brute-force search can be kept linear if the scheme intFC one starts from has the preprocessing property. The observation is that all the np verifications are done with respect to the same function. Thus one can first compute $\mathsf{vk}_{\boldsymbol{f}}$ using the preprocessing algorithm, in time $O_\lambda(n)$ ($O_\lambda(n \log \lambda)$ for $p = O(\log \lambda)$), and then run the np verifications, each in fixed n-independent time $O_\lambda(1)$. Hence, using preprocessing we can achieve a verification that is $O_\lambda(pn)$, which is $O_\lambda(n)$ (or $O_\lambda(n \log \lambda)$) for small domains $p = O(1)$

(or $p = O(\log \lambda)$ resp.). As a drawback, one can notice that this brute-force search inherently loses the possibility of achieving preprocessing verification.

We conclude observing that, by considering an instantiation of intFC obtain by applying the transformation of Section 4.2 to the FC of Section 3 we obtain, for $p = O(1)$ ($p = O(\log \lambda)$), an FC for inner products $\pmod p$ in which openings have fixed size $O_\lambda(1)$ and verification is $O_\lambda(n)$ ($O_\lambda(n \log \lambda)$ resp.).

5.2 A variant of our FC for binary inner products mod p

In this section, we present another approach of our FC in 3 to treat binary inner products mod p. That is, we let the prover computes g to the power of $L(y), R(y)$ for the verifier. The add-ons to the opening proof is the g to the power of $L(y), R(y)$ accompanied with an argument of knowledge showing that the integer y encoded in these exponentiation is equivalent to y_p modulo p.

Intuition Here we explain the initial idea of our argument of knowledge. For simplicity, we first consider p to be a power of 2, in particular, $p = 2^\ell$. Then we can express $y = q2^\ell + y_p$ with some quotient q and residue y_p. Recall from fig. 2 that $L(y) = 2^{\kappa y}$ and $R(y) = 2^{(\kappa + \frac{\kappa}{n})y}$. Here we are ready to describe the hints.

The prover should send $Q_L = g^{2^{\kappa q}}$, $Q_R = g^{2^{(\kappa+\frac{\kappa}{n})q}}$ and also $Q_{L,j} = g^{2^{\kappa q 2^j}}$ and $Q_{R,j} = g^{2^{(\kappa+\frac{\kappa}{n})q2^j}}$ for $j = 1, \ldots, \ell$ to the verifier.

She also uses a Proof of Square Exponent PoSE for each consecutive pair $Q_{L,j}, Q_{L,j+1}$, to show that each pair is of the form g^x, g^{x^2} for the known base g and some x. Similarly, she also needs to prove this relation for each consecutive pair $Q_{R,j}, Q_{R,j+1}$

At the end, the verifier checks the validity of the PoSE's and also checks that $Q_{L,\ell}^{2^{\kappa y_p}} = g^{L(y)}$ and $Q_{R,\ell}^{2^{(\kappa+\frac{\kappa}{n})y_p}} = g^{R(y)}$.

However, the modulo p could be known before $g^{L(y)}$ and $g^{R(y)}$ are fixed. In order to turn this idea into an argument of knowledge, we let the verifer send another prime modulo e as a challenge. Moreover, to treat generalized modulo p that are not powers of 2, we merge the construction with a square-and-multiply algorithm.

PoKEEM. The proof of knowledge of exponent of exponent modulo p (PoKEEM) is for the following relation

$$\mathcal{L}_{\mathsf{PoKEEM}} = \left\{ (Y, a, p, x_p; x) \in \mathbb{G} \times \mathbb{Z} : Y = g^{a^x}, x = x_p \bmod p \right\}$$

parametrized by a group $\mathbb{G} \leftarrow\!\$\ \mathsf{Ggen}(\lambda)$ and a group element $g \leftarrow\!\$\ \mathbb{G}$. The protocol is in Figure 4.

$\mathsf{Setup}(\lambda)$: $\mathbb{G} \leftarrow\!\!\$\ \mathsf{Ggen}(\lambda)$, $g \leftarrow\!\!\$\ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{V} \rightarrow \mathsf{P}}$: Sends $e \leftarrow\!\!\$\ \mathsf{Primes}(\lambda)$
Denote the binary representation $ep = \overline{b_s \cdots b_1}$ and $e_i = \lfloor e/2^{s-i} \rfloor$;
$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Computes $q = \lfloor x/e \rfloor$, $r = x \bmod ep$ and sets $Q_i = g^{a^{e_i q}}$ for each $i \in [1, s]$ and $Q'_i = g^{a^{2e_i q}}$ for each $i \in [1, s-1]$. For each $i \in [1, s-1]$, computes $\pi^{(i)}_{\mathsf{PoSE}} = \mathsf{PoSE.P}(\mathsf{crs}, (Q_i, Q'_i), a^{e_i q})$ and $\pi^{(i)}_{\mathsf{PoDDH}} = \mathsf{PoDDH.P}(\mathsf{crs}, (Q_i, Q'_i, Q_{i+1}), (a^{e_i q}, a^{2e_i q}))$ if $b_{i+1} = 1$ or $\pi^{(i)}_{\mathsf{PoDDH}} = (Q_{i+1} = Q'_i)$ if $b_{i+1} = 0$.
Sends $\Lambda := (\boldsymbol{Q}, \boldsymbol{Q'}, \boldsymbol{\pi}, r)$ for $\boldsymbol{Q} = (Q_1, \ldots, Q_s), \boldsymbol{Q'} = (Q'_1, \ldots, Q_{s-1}), \boldsymbol{\pi} = (\pi^{(i)}_{\mathsf{PoSE}})_{i=1}^{s-1}$ and $\boldsymbol{\pi'} = (\pi^{(i)}_{\mathsf{PoDDH}})_{i=1}^{s-1}$.
$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \Lambda)}$: Outputs 1 iff $r \in [ep]$, $x_p = r \bmod p$, ${Q_s}^{a^r} = Y$ and all the proofs $\boldsymbol{\pi}, \boldsymbol{\pi'}$ verify.

Fig. 4: The succinct argument of knowledge of exponent (PoKEEM) protocol.

Theorem 4. *The protocol* PoKEEM *is an argument of knowledge for the language* $\mathcal{L}_{\mathsf{PoKEEM}}$.

Proof. (Sketch) The proof starts from running the extractor for every proof $\pi^{(i)}_{\mathsf{PoSE}}$ and $\pi^{(i)}_{\mathsf{PoSE}}$ in $\boldsymbol{\pi}, pi'$ to obtain a list of witnesses. These witnesses are the z_i's and z'_i's such that $g^{z_i} = Q_i$ and $g^{z'_i} = Q'_i$. The subsequent argument is to show that these exponent values are consistent (i.e. we should not extract two distinct values for some z_i coming from different proofs). This holds indeed, otherwise we find some low-order relation $g^c = 1$ for c is the difference of the two values. In particular, $z_s = epz_1$.

Now, simplifying the notation $z = z_1$, we want to show that z is of the form a^x for some integer x. Rewinding the protocol for another challenge e', we obtain another response r' and extracted value z' such that $Y = g^{z^{ep} a^r} = g^{z^{e'p} a^{r'}}$. We argue that $z^{ep} a^r = z'^{e'p} a^{r'}$, otherwise we find a low-order relation. Now we represent $z = da^c$ and $z' = d'a^{c'}$ such that $d, d' \nmid a$. Therefore, we have $d^{ep} a^{cep - c'e'p + r - r'} = d'^{e'p}$. W.l.o.g we may assume that $t = cep - c'e'p + r - r'$ is positive, and the relation $d^{ep} a^t = d'^{e'p}$ is over $\mathbb{N}$. By some number-theoretical arguments together with an observation that d, d' are chosen before e, e' are sampled and also $(e, e') = 1$ with overwhelming probability, we derive that $t = 0$ and $d = d' = 1$. In other words, $z = a^c$ and $Y = g^{a^{cep + y_p}}$.

Our variant We are ready to describe our FC for binary inner products mod p more precisely. We only describe the differences with the scheme of Section 3.

In the Open algorithm, the prover additionally includes the values $g^{L(y)}, g^{R(y)}$ in the opening proof Λ. She also plugs the PoKEEM proof, computed for $a = 2^\kappa$ and $a = 2^{(\kappa + \frac{\kappa}{n})}$, in order to prove that the exponents of exponents in $g^{L(y)}$ and $g^{R(y)}$ are congruent to y_p, respectively.

In the Ver algorithm, in order to verify an output y_p, the verifier first runs the verification of Section 3 with the difference that it checks the range proof by using the values $g^{L(y)}, g^{R(y)}$ included in Λ. Next, the verifier checks the validity of the PoKEEM proof for elements $g^{L(y)}, g^{R(y)}$ and exponent y_p.

Security We state the theorem below to fulfill the security of our FC variant over $\mathbb{Z}_p$.

Theorem 5. *If protocol* PoKEEM *is an argument of knowledge for* $\mathcal{L}_{\mathsf{PoKEEM}}$ *and our binary functional vector commitment is functional binding, then our variant of binary functional vector commitment over* $\mathbb{Z}_p$ *is functional binding.*

The proof directly comes from the extraction of an integer y congruent to y_p modulo p from $g^{L(y)}, g^{R(y)}$ and an observation that the existence two valid opening modulo p immediately implies the existence of two opening over integer.

Efficiency This modification adds up the opening size for binary inner products an $O_\lambda(\log p + \lambda)$ complexity. It also takes time $O_\lambda(\log p + \lambda) + t_{\mathsf{Ver}}(n)$ to verify, where $t_{\mathsf{Ver}}(n)$ is the verification time of intFC.

Considering an instantiation of intFC deriving from applying the transformation of Section 4.2 to the FC of Section 3 we then, for $p = O(1)$, obtain an FC for inner products modulo p in which openings are of size $O_\lambda(1)$ and verification is in time $O_\lambda(1)$ with preprocessing.

Acknowledgements The second and third authors received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), by the Spanish Government under projects SCUM (ref. RTI2018-102043-B-I00) and RED2018-102321-T, and by the Madrid Regional Government under project BLOQUES (ref. S2018/TCS-4339). This work is also supported by a research grant (ref. PL-RGP1-2021-051) from Protocol Labs.

References

AGL+22. A. Arun, C. Ganesh, S. Lokam, T. Mopuri, and S. Sridhar. Dew: Transparent Constant-sized zkSNARKs. Cryptology ePrint Archive, Report 2022/419, 2022.

AHIV17. S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

BBB+18. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

BBF18. D. Boneh, B. Bünz, and B. Fisch. A Survey of Two Verifiable Delay Functions. Cryptology ePrint Archive, Report 2018/712, 2018.

BBF19. D. Boneh, B. Bünz, and B. Fisch. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

BBHR19. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable Zero Knowledge with No Trusted Setup. In *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, Heidelberg, August 2019.

BCC+16. J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

BCR+19. E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent Succinct Arguments for R1CS. In *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

Bd94. J. C. Benaloh and M. de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract). In *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285. Springer, Heidelberg, May 1994.

BFS20. B. Bünz, B. Fisch, and A. Szepieniec. Transparent SNARKs from DARK Compilers. In *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.

BH01. J. Buchmann and S. Hamdy. A Survey on {IQ} Cryptography, 2001.

BP97. N. Bari and B. Pfitzmann. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 480–494. Springer, Heidelberg, May 1997.

BS96. W. Bosma and P. Stevenhagen. On the computation of quadratic 2-class groups. *Journal de théorie des nombres de Bordeaux*, 8(2):283–313, 1996.

CF13. D. Catalano and D. Fiore. Vector Commitments and Their Applications. In *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Heidelberg, February / March 2013.

CFG+20. M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, and L. Nizzardo. Incrementally Aggregatable Vector Commitments and Applications to Verifiable Decentralized Storage. In *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 3–35. Springer, Heidelberg, December 2020.

CMS99. C. Cachin, S. Micali, and M. Stadler. Computationally Private Information Retrieval with Polylogarithmic Communication. In *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 402–414. Springer, Heidelberg, May 1999.

COS20. A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and Transparent Recursive Proofs from Holography. In *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Heidelberg, May 2020.

CS99. R. Cramer and V. Shoup. Signature Schemes Based on the Strong RSA Assumption. In *ACM CCS 99*, pages 46–51. ACM Press, November 1999.

FS87. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

GHR99. R. Gennaro, S. Halevi, and T. Rabin. Secure Hash-and-Sign Signatures Without the Random Oracle. In *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 123–139. Springer, Heidelberg, May 1999.

GKM+18. J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. Updatable and Universal Common Reference Strings with Applications to zk-

SNARKs. In *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018.

Gro05. J. Groth. Non-interactive Zero-Knowledge Arguments for Voting. In *ACNS 05*, volume 3531 of *LNCS*, pages 467–482. Springer, Heidelberg, June 2005.

KZG10. A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

Lip03. H. Lipmaa. On Diophantine Complexity and Statistical Zero-Knowledge Arguments. In *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 398–415. Springer, Heidelberg, November / December 2003.

Lip12. H. Lipmaa. Secure Accumulators from Euclidean Rings without Trusted Setup. In *ACNS 12*, volume 7341 of *LNCS*, pages 224–240. Springer, Heidelberg, June 2012.

LM19. R. W. F. Lai and G. Malavolta. Subvector Commitments with Application to Succinct Arguments. In *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 530–560. Springer, Heidelberg, August 2019.

LP20. H. Lipmaa and K. Pavlyk. Succinct Functional Commitment for a Large Class of Arithmetic Circuits. In *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 686–716. Springer, Heidelberg, December 2020.

LRY16. B. Libert, S. C. Ramanna, and M. Yung. Functional Commitment Schemes: From Polynomial Commitments to Pairing-Based Accumulators from Simple Assumptions. In *ICALP 2016*, volume 55 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl, July 2016.

LY10. B. Libert and M. Yung. Concise Mercurial Vector Commitments and Independent Zero-Knowledge Sets with Short Proofs. In *TCC 2010*, volume 5978 of *LNCS*, pages 499–517. Springer, Heidelberg, February 2010.

Mic94. S. Micali. CS Proofs (Extended Abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

PPS21. C. Peikert, Z. Pepin, and C. Sharp. Vector and Functional Commitments from Lattices. In *Theory of Cryptography Conference*, pages 480–511. Springer, 2021.

Set20. S. Setty. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020.

SL20. S. Setty and J. Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020.

Wes18. B. Wesolowski. Efficient verifiable delay functions. Cryptology ePrint Archive, Report 2018/623, 2018. `https://eprint.iacr.org/2018/623`.

Wes19. B. Wesolowski. Efficient Verifiable Delay Functions. In *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

WTs+18. R. S. Wahby, I. Tzialla, a. shelat, J. Thaler, and M. Walfish. Doubly-Efficient zkSNARKs Without Trusted Setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

ZXZS20. J. Zhang, T. Xie, Y. Zhang, and D. Song. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *2020 IEEE Symposium on Security and Privacy*, pages 859–876. IEEE Computer Society Press, May 2020.

A Argument of Knowledge Protocols

Protocol PoE: On top of the succinct arguments of knowledge described in Section 2.3, to optimize our constructions' verification time we can utilize the PoE protocol, introduced by Wesolowski [Wes19] for exponents of the form 2^T and generalized by [BBF19] for arbitrary exponents. That is a sound argument for the relation:

$$\mathcal{L}_{\mathsf{PoE}} = \left\{ \left(Y, u, x \in \mathbb{G}^2 \times \mathbb{Z}; \varnothing\right) : Y = u^x \right\}$$

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow\!\!\$\ \mathsf{Ggen}(\lambda)$, $g \leftarrow\!\!\$\ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{V} \rightarrow \mathsf{P}}$: Sends $\ell \leftarrow\!\!\$\ \mathsf{Primes}(\lambda)$
$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Computes $q = \lfloor x/\ell \rfloor$ and sets $Q = u^q$. Sends $\pi := Q$.
$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Computes $r = x \mod \ell$ and outputs 1 iff $Q^\ell u^r = Y$.

Fig. 5: The succinct sound argument (PoE).

PoE is a sound argument system under the adaptive root assumption for Ggen and it is not knowledge sound, since there is no witness. The verifier knows the exponent x. It is used for to improve verification: the verifier performs an $O(\lambda)$ group exponentiation and $O(\|x\|)$ integer operations, instead of an $O(\|x\|)$ group exponentiation. Although asymptotically this is the same, concretely it gives a significant improvement, since integer operations are much more efficient. Furthermore, its size is 1 group element independently of the size of x.

The proof of knowledge of exponent.

$$\mathcal{L}_{\mathsf{PoKE}} = \{(Y, u; x) \in \mathbb{G} \times \mathbb{Z} : Y = u^x\}$$

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow\!\!\$\ \mathsf{Ggen}(\lambda)$, $g \leftarrow\!\!\$\ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Sends $z = g^x$
$\underline{\mathsf{V} \rightarrow \mathsf{P}}$: Sends $\ell \leftarrow\!\!\$\ \mathsf{Primes}(\lambda)$
$\underline{\mathsf{P} \rightarrow \mathsf{V}}$: Computes $q = \lfloor x/\ell \rfloor$, $r = x \mod \ell$ and sets $Q = u^q, Q' = g^q$. Sends $\pi := (Q, Q', r)$.
$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Outputs 1 iff $r \in [\ell]$, $Q^\ell u^r = Y$ and $Q'^\ell g^r = z$.

Fig. 6: The succinct argument of knowledge of exponent (PoKE) protocol.

Remark 4. The above protocol is for arbitrary bases u, adversarially chosen. As noted in [BBF19] the protocol gets simpler in case the base is trusted, generated

in the setup phase (i.e. $u = g$ and $Y = g^x$). In particular, z shall not be sent (since it's the statement, $z = Y$) and the proof is 1 group element less.

The proof of Diffie-Hellman tuple.

$$\mathcal{L}_{\mathsf{PoDDH}} = \left\{(Y_0, Y_1, Y; x_0, x_1) \in \mathbb{G}^3 \times \mathbb{Z}^2 : g_0^{x_0} = Y_0 \wedge g_1^{x_1} = Y_1 \wedge g^{x_0 x_1} = Y\right\}$$

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow\!\$\ \mathsf{Ggen}(\lambda)$, $g, g_0, g_1 \leftarrow\!\$\ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{V} \to \mathsf{P}}$: Sends $\ell \leftarrow\!\$\ \mathsf{Primes}(\lambda)$
$\underline{\mathsf{P} \to \mathsf{V}}$: Computes $q = \lfloor x_0 x_1/\ell \rfloor$, $q_0 = \lfloor x_0/\ell \rfloor$, $q_1 = \lfloor x_1/\ell \rfloor$, $r_0 = x_0 \mod \ell$, $r_1 = x_1 \mod \ell$ and sets $Q = g^q$, $Q_0 = g_0^{q_0}$, $Q_1 = g_1^{q_1}$. Sends $\pi := (Q, Q_0, Q_1, r_0, r_1)$.
$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Computes $r = r_0 r_1 \mod \ell$. Outputs 1 iff $Q^\ell g^r = Y$ and $Q_0^\ell g^{r_0} = Y_0$ and $Q_1^\ell g_1^{r_1} = Y_1$.

Fig. 7: The succinct argument of knowledge of Diffie-Hellman tuple (PoDDH), under different bases g, g_0, g_1.

The succinct proof of square exponent.

$$\mathcal{L}_{\mathsf{PoSE}} = \left\{(Z_i; x_i) \in \mathbb{G} \times \mathbb{Z} : g^{x_i^2} = Z_i\right\}.$$

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow\!\$\ \mathsf{Ggen}(\lambda)$, $g \leftarrow\!\$\ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{P} \to \mathsf{V}}$: Sends $z_i = g^{x_i}$
$\underline{\mathsf{V} \to \mathsf{P}}$: Sends $\ell \leftarrow\!\$\ \mathsf{Primes}(\lambda)$
$\underline{\mathsf{P} \to \mathsf{V}}$: Computes $q = \lfloor x_i/\ell \rfloor$, $r = x_i \mod \ell$ and sets $Q = z_i^q, Q' = g^q$. Sends $\pi := (Q, Q', r)$.
$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Outputs 1 iff $r \in [\ell]$, $Q^\ell z_i^r = Z_i$ and $Q'^\ell g^r = z_i$.

Fig. 8: The succinct argument of knowledge of square exponent (PoSE).

The succinct range proof protocol

$$\mathcal{L}_{\mathsf{PoRE}} = \{(Y, L, R; x) \in \mathbb{G} \times \mathbb{Z} : L < x < R \wedge g^x = Y\}.$$

$\mathsf{Setup}(\lambda) : \mathbb{G} \leftarrow\!\!\$\ \mathsf{Ggen}(\lambda)$, $g, g_0, g_1 \leftarrow\!\!\$\ \mathbb{G}$, set $\mathsf{crs} := (\mathbb{G}, g)$.

$\underline{\mathsf{P} \to \mathsf{V}}$: Computes:
- $Y' = g^{(x-L)(R-x)}$
- For $i = 1, 2, 3$: $Z_{x_i} = g_1^{x_i^2}$, $z_{x_i} = g_1^{x_i}$

Sends $(Y', \{Z_{x_i}, z_{x_i}\}_{i=1}^3)$

$\underline{\mathsf{V} \to \mathsf{P}}$: Sends $\ell \leftarrow\!\!\$\ \mathsf{Primes}(\lambda)$

$\underline{\mathsf{P} \to \mathsf{V}}$: Computes:
- $q = \lfloor x/\ell \rfloor$, $Q_b = g^q$, $r = x \mod \ell$.
- For $i = 1, 2, 3$: $q_{x_i} = \lfloor x_i/\ell \rfloor$, $Q_{x_i} = z_i^{q_{x_i}}$, $Q'_{x_i} = g_i^{q_{x_i}}$, $r_{x_i} \mod \ell$
- $q_L = \lfloor (x-L)/\ell \rfloor$, $Q_L = g_1^{q_L}$, $r_L = (x-L) \mod \ell$
- $q_R = \lfloor (R-x)/\ell \rfloor$, $Q_R = g_1^{q_R}$, $r_R = (R-x) \mod \ell$
- $q_{LR} = \lfloor (x-L)\cdot(R-x)/\ell \rfloor$, $Q_{LR} = g_1^{q_{LR}}$, $r_{LR} = (x-L)\cdot(R-x) \mod \ell$

Sends $\pi := \big(Q, r, \{Q_{x_i}, Q'_{x_i}, r_{x_i}\}_{i=1}^3, Q_L, r_L, Q_R, r_R, Q_{LR}\big)$.

$\underline{\mathsf{V}(\mathsf{crs}, Y, \ell, \pi)}$: Outputs 1 iff:
- $Q^\ell g^r = Y$
- $Q_{x_i}'^\ell g^{r_{x_i}} = z_i$, $Q_{x_i}^\ell z_{x_i}^{r_{x_i}} = Z_{x_i}$, for $i = 1, 2, 3$
- $Q_L^\ell g^{r_L} = Y \cdot g^{-L}$, $Q_R^\ell g^{r_R} = g^R \cdot Y^{-1}$, $Q_{LR}^\ell g^{r_{LR}} = Y'$
- $Y'^4 \cdot g = \prod_{i=1}^3 Z_i$

where $r = r_0 r_1 \mod \ell$ and $r_{LR} = r_L r_R \mod \ell$.

Fig. 9: The succinct Argument of Knowledge of range of an exponent.