Efficient UC Commitment Extension with Homomorphism for Free (and Applications)

Ignacio Cascudo[1*], Ivan Damgård[2†], Bernardo David[3‡], Nico Döttling[4], Rafael Dowsley[5†], and Irene Giacomelli[6§]

[1] IMDEA Software Institute, Madrid, Spain, ignacio.cascudo@imdea.org
[2] Aarhus University, Aarhus, Denmark, ivan@cs.au.dk
[3] IT University of Copenhagen, Copenhagen, Denmark, bernardo@bmdavid.com
[4] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany, nico.doettling@gmail.com
[5] Bar Ilan University, Tel Aviv, Israel, rafael@dowsley.net
[6] Protocol Labs, Inc., Basel, Switzerland, irene@protocol.ai

Abstract. Homomorphic universally composable (UC) commitments allow for the sender to reveal the result of additions and multiplications of values contained in commitments without revealing the values themselves while assuring the receiver of the correctness of such computation on committed values. In this work, we construct essentially optimal additively homomorphic UC commitments from any (not necessarily UC or homomorphic) extractable commitment, while the previous best constructions require oblivious transfer. We obtain amortized linear computational complexity in the length of the input messages and rate 1. Next, we show how to extend our scheme to also obtain multiplicative homomorphism at the cost of asymptotic optimality but retaining low concrete complexity for practical parameters. Moreover, our techniques yield public coin protocols, which are compatible with the Fiat-Shamir heuristic. These results come at the cost of realizing a restricted version of the homomorphic commitment functionality where the sender is allowed to perform any number of commitments and operations on committed messages but is only allowed to perform a single batch opening of a number of commitments. Although this functionality seems restrictive, we show that it can be used as a building block for more efficient instantiations of recent protocols for secure multiparty computation and zero knowledge non-interactive arguments of knowledge.

---

*This work was done while Ignacio Cascudo was with Department of Mathematics, Aalborg University, Denmark.

†This project has received funding from the European Research Council (ERC) under the European Unions' Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO).

‡This work was partially supported by DFF grant number 9040-00399B (TrA^{2}C).

§This work was done while Irene Giacomelli was with the ISI Foundation (Turin, Italy) and supported by Intesa Sanapolo Innovation Center.

1 Introduction

A commitment scheme is the digital equivalent of a locked box containing a committed message chosen by a prover. Once the prover gives away the box to a verifier, the content cannot be changed, the commitment is binding. On the other hand, the verifier cannot look into the box so the message is hidden until the prover gives away the key to the box. Commitments are perhaps the most fundamental building block in cryptographic protocols and despite the conceptual simplicity of the primitive, it has far-reaching consequences and many applications, e.g., to coin-flipping, zero-knowledge proofs and many other things.

The simplest form of commitment that only have the basic binding and hiding properties follow from one-way functions. On the other hand, one may wish for many other properties, such as non-malleability, security under composition etc. The strongest form of commitments, namely UC secure commitments, has all these properties, but on the other hand can only be implemented under setup assumptions, such as the common reference string model. In this model, UC commitments imply secure key exchange, so since some sort of public-key technology seems to be required, it was believed for a long time that even if UC commitments are the gold standard for security, they must be much less efficient than the weaker type that only requires symmetric primitives.

However, in [19] and independently in [24,9], this was shown to be false: one can push the use of public-key technology into a preprocessing phase that is only needed once and for all and the cost of which does not depend on the number of commitments to be done later. Notably, the actual commitment and opening protocols only requires simple finite field algebra and a pseudorandom generator. After this, a long line of research optimized this approach [17,22], culminating in [16] where it was shown that after doing $O(k+s)$ string OTs in the setup phase (where s is the statistical security parameter and k is the message length) one can commit at rate approaching 1, that is, the communication required is $k + o(1)$ bits, furthermore the computational complexity is linear in k[7]. Finally, the commitments are additively homomorphic, i.e., one commits to vectors over a finite field $\mathbb{F}$, and if $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^k$ have been committed, prover and verifier can compute a commitment to $\boldsymbol{a} + \boldsymbol{b}$ which, if opened, would reveal only the sum.

The first construction from this line of work [19] had also a multiplicatively homomorphic property, namely the prover can send the verifier a single message, and this allows the verifier to compute a commitment to $\boldsymbol{a} * \boldsymbol{b}$, the coordinate-wise (Schur) product of the vectors. However, subsequent constructions did not have this property.

So, while this line of research has resulted in constructions that are optimal in several respects, it still leaves some important and natural questions unanswered:

Is it overkill to use OT in the setup phase? All efficient earlier schemes [19,24,17,22,16] use OT in the preprocessing phase, but this is in general

[7] All this holds in an amortized sense, assuming we make enough commitments so that the cost of the setup phase is dwarfed.

a stronger primitive than commitment. Even UC commitments do not always imply OT, this depends on the setup assumption. It is therefore natural to ask if we can make do with only commitment in the preprocessing, thus obtaining a proper "commitment extension" result.

Can we make an efficient multi-verifier scheme? The commitments from [16], and in fact all constructions from this line of work, can only work with one verifier because security against a corrupt prover depends on the verifier's private choice of selections bits in the initial OT's. Thus, if a prover needs to commit towards several verifiers, the only known solution is to run many instances of the scheme, one for each verifier and then on top of this have the prover convince the verifiers that (s)he committed to the same message. This seems quite far from an ideal solution.

Can we also get multiplicatively homomorphic schemes? The most efficient constructions are not multiplicative, but one earlier scheme was in fact "fully homomorphic" [19]. So it is natural to ask if we can solve the above problems and also get multiplication at the same time.

1.1 Our contributions

In this paper, we come up with positive answers to all of the above questions. We present a protocol for UC secure commitments that has the well known structure consisting of a preprocessing phase and a phase where the actual commitments are built, computed on and opened. In addition to achieving the same asymptotic efficiency as the former best scheme [16] in the single-verifier additive case, our protocol supports multiple verifiers and multiplicative homomorphism.

In contrast to previous work, however, the preprocessing only makes use of a commitment scheme (and not OT)[8]. Notably, however, this commitment scheme does not need to be homomorphic, and in fact it does not even need to be UC secure. It just needs to be extractable and hiding - here, extractable means that the simulator can extract the committed value from a corrupt prover. For UC full security one usually needs also equivocation (when the prover is honest, the simulator can fake a commitment and later open it to any value). The commitment scheme we build uses only a PRG and finite field arithmetic after the preprocessing. It has rate 1, it is additively homomorphic, and linear time. Security does not depend on any secret choices of the verifier, so the scheme easily extends to multiple verifiers with no essential loss of security. Finally, we show how to make the scheme multiplicative, the scheme is then only quasilinear, and we get constant rate instead of rate 1.

All these results come at the cost that what we implement is a slightly weaker commitment functionality than the standard one. Namely, it allows opening of committed values only in a final stage and after this the functionality stops working. Equivalently, one can think of this as a functionality one can use exactly

[8]The scheme of [9] can be constructed from an extractable commitment and an equivocal commitment. However, it is intrinsically incompatible with homomorphic operations.

as the standard one, except that when opening a value the prover simply tells the verifier what the committed value is. Of course a corrupt prover can lie, but there is a final verification stage where the prover will be caught if he lied.

We show that despite this limitation there are a wide range of applications for the scheme. While we describe these in more detail below, it is already intuitively clear that our functionality is sufficient for ZK proofs, for instance: the verifier needs to decide to accept or reject only at the end of the protocol so it is sufficient that a cheating prover is caught at that point. As a simple example of the power of our construction, consider that UC secure commitments are easy to implement in the (global) random oracle model [11]: one simply inputs the message concatenated with some randomness to the oracle and uses the output as the commitment. Of course, a random oracle based scheme has no homomorphic properties: a random oracle "by definition" has no such structure. But nevertheless, we can use it as commitment scheme in our preprocessing and get a homomorphic scheme. In general, one can think of our protocol as a "commitment extension" result. It is similar to the well known OT extension protocols, but incomparable because we get extra homomorphic properties (and perhaps UC security) for free, but we realize a slightly weaker functionality.

Techniques. On the technical side, our approach is best described by referring to previous work such as [16]: the main idea there was that the prover commits to a vector $\boldsymbol{a}$ by encoding it using a linear code C. He then additively secret shares each coordinate in the codeword $\mathsf{C}(\boldsymbol{a})$ to get two shares for each position. Using the OT's from the preprocessing, the verifier will learn one out of the two shares for each position, however, the prover does not know which shares the verifier has. To open, the prover must reveal $\mathsf{C}(\boldsymbol{a})$ and all shares, and the verifier can now check that the prover sent a codeword and that the shares are consistent with $\mathsf{C}(\boldsymbol{a})$ and with the shares the verifier knows.

Intuitively, since the verifier has only one share of each coordinate, $\mathsf{C}(\boldsymbol{a})$ is unknown to him at commit time. On the other hand, if the prover wants to open a different value, he must change to a different codeword. However, if C has large minimum distance, this means the prover must change many coordinates and therefore must lie about many of the shares. Since he does not know which shares he can change without being detected, this can only be done with negligible success probability[9].

In order to avoid having to do an OT for each codeword position and each commitment, instead the prover chooses seeds $\boldsymbol{s}_{i,j}$ for a PRG, where i points to a codeword position and $j = 0, 1$. The shares for all the commitments are then constructed by running the PRG on all these seeds and for each i an OT is done that transfers either $\boldsymbol{s}_{i,0}$ or $\boldsymbol{s}_{i,1}$ to the verifier.

Our key observation now is that it is actually sufficient if the prover simply commits to the seeds in the preprocessing phase, if we are careful later. Namely,

[9]This argument works, even if the prover did not choose a codeword at commit time. If we also want to have additive homomorphism, we need to check that the prover chose something that it at least close to a codeword. This can be done using, e.g., the interactive proximity testing from [16].

we run the same protocol as we would have done had the OTs been used, but at the end of the protocol, the verifier will ask the prover to reveal either $\boldsymbol{s}_{i,0}$ or $\boldsymbol{s}_{i,1}$ for each i. Note that, as long as a corrupt prover cannot predict which seeds he will be asked for, he is in the exactly same position as in the original protocol. The verifier will receive the same information as before, but cannot verify it until the end, so hence openings can only be done, or at least can only be verified, at the end. A corrupt verifier clearly has no advantage compared to the OT based protocol: he learns the same information, only later.

A very nice "side effect" of this is that we can now easily have several verifiers. They just need to receive the prover's initial commitments (assuming, of course that the initial commitments support this). Then at the end, they can decide, e.g., by coin flipping which seeds to ask for.

We also extend the commitment scheme to allow for proving multiplicative relations on committed values. For this purpose, we require the code C to have the property that its square C^{*2} is also a good code, with large minimum distance. Here C^{*2} is defined to be the span of all pairwise Schur-products of words from C. Moreover, we replace the 2-party additive secret sharing by 3-party linear secret sharing which is multiplicative: the Schur-product of sets of shares of $u, v \in \mathbb{F}$ is (essentially) an additive secret sharing of uv. The effect of all this is that if we multiply two commitments to $\boldsymbol{a}, \boldsymbol{b}$ by multiplying corresponding components of them, we obtain a commitment to $\boldsymbol{a} * \boldsymbol{b}$ of essentially the same form as in the original protocol, except that underlying code is now C^{*2}. See more details within. The new demands we place on C imply that we can only get constant rate and not rate 1 and also that complexity will be quasilinear rather than linear. The main motivation for this construction is that we get the multiplicative property and at the same time have multiple verifiers and use only commitment for preprocessing. An earlier scheme that achieves multiplicative homomorphism was constructed in [19] via building first an elaborate VSS (verifiable secret sharing) scheme. Our construction obtains similar asymptotic complexity, but it requires less conditions on the underlying linear code. Indeed, our multiplicatively homomorphic scheme can be constructed from any linear code whose minimum distance and squares minimum distance are large enough. In contrast, [19] requires in addition a code whose duals minimum distance is large enough (*i.e.*, equivalent to multiplicative secret sharing scheme). Thanks to this, for fixed security parameters we can give an explicit bound for the rate of our multiplicative commitment based on recent results on squares of cyclic codes (details in Section 4).

1.2 Applications

Efficient Zero-Knowledge Arguments. A recent line of research is concerned with the construction of practically efficient succinct non-interactive zero-knownledge arguments of knowledge (e.g. [1,10,31]) with a particular focus on optimizing the efficiency of the prover while keeping verification complexity sub-linear.

One such approach, originally dating back to [5], compiles a public coins interactive proof system for a language $\mathcal{L}$ into a zero-knowledge proof system for

the same language. This transformation is conceptually simple: Instead of sending its messages to the verifier in the clear, the prover provides only commitments of his messages to the verifier. At the end of the protocol, the prover provides a zero-knowledge proof to the verifier which asserts that the verifier of the original proof system would accept the committed transcript. This transformation has received renewed interest in the light of efficient P-delegation schemes [25,29].

Wahby et al. [31] observed that this approach can be implemented in a particularly efficient way if the verifier of the interactive proof system is algebraic: In this case the zero-knowledge proof in the transformation of [5] can be implemented very efficiently via homomorphic commitments.

We show that using our homomorphic commitment scheme, this transformation can be performed at a very low overhead, i.e. we can convert any public coin interactive proof system with algebraic verifier into an honest-verifier zero-knowledge proof system such that the communication complexity of the protocol grows only by a small factor and both prover and verifier incur only a small constant factor overhead. Using the Fiat Shamir transform [20], we can convert such a proof system into a succinct non-interactive zero-knowledge argument.

Committed MPC. The so called "Committed MPC" protocol [21] requires a multiparty additively homomorphic commitment protocol that supports additions of commitments generated by different senders. While a generic approach for constructing such schemes from any two-party additively homomoprhic commitments was proposed in [21], their generic construction for t parties requires t^2 calls to the underlying commitment scheme. If instantiated with the previously best two-party additively homomorphic commitment protocol of [16] using a $[n, k, s]$ code, this construction would require nt^2 OTs plus extra communication in the order of $O(nmt^2)$ to commit to m messages of length k. We provide a new generic construction from multi-receiver additively homomorphic commitments which can be instantiated with our new protocols, requiring only nt non-homomorphic commitments (*e.g.* random oracle commitments) plus extra communication in the order of $O(smt)$ to achieve the same.

Insured MPC. The topic of MPC with financial penalties has attracted increasing attention recently [2,6,26,7,4]. The main idea is to combine MPC techniques with cryptocurrencies in order to provide monetary incentives for the participants to act honestly during the protocol execution. Insured MPC [4], the most efficient solution to date, uses a publicly verifiable additively homomorphic multi-receiver commitment as an important component to build the protocol. However, the employed commitment scheme is a bottleneck in that construction as its complexity grows quadratically in the number of participants. Using our new techniques together with an authenticated bulletin board (which is also used in the previous construction), it is possible to dramatically improve the performance of publicly verifiable additively homomorphic multi-receiver commitment. We can obtain extremely efficient instantiations, for instance, by using the canonical random oracle commitment scheme. The improvement in computational and communication complexity achieved for this application is very similar to that

of the Committed MPC case, since the previously best protocol for publicly verifiable additively homomorphic multi-receiver commitments [4] has a very similar structure to the multi-sender protocol of [21]. Thus, we basically go from quadratic to linear in the number of players.

2 Preliminaries

In this section we establish notation and introduce notions that will be used throughout the paper. We borrow much of the notation from [16].

Notation. The set of the n first positive integers is denoted $[n] = \{1, 2, \ldots, n\}$. Given a finite set D, sampling a uniformly random element from D is denoted $r \xleftarrow{\$} D$. Vectors of elements of some field are denoted by bold lower-case letters, while matrices are denoted by bold upper-case letters. We denote finite fields by $\mathbb{F}$ and write $\mathbb{F}_q$ for the finite field of size q. For $\boldsymbol{z} \in \mathbb{F}^k$, $\boldsymbol{z}[i]$ denotes the i'th entry of the vector, where $\boldsymbol{z}[1]$ is the first element of $\boldsymbol{z}$. The coordinate-wise (Schur) product of two vectors is denoted by $*$, *i.e.* if $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$, then $\boldsymbol{a} * \boldsymbol{b} \in \mathbb{F}^n$ and $(\boldsymbol{a} * \boldsymbol{b})[i] = \boldsymbol{a}[i]\boldsymbol{b}[i]$. If $A \subseteq [n]$, we will use π_A to denote the projection that outputs the coordinates with index in A of a vector. For a matrix $\mathbf{M} \in \mathbb{F}^{n \times k}$, we let $\mathbf{M}[\cdot, j]$ denote the j'th column of $\mathbf{M}$ and $\mathbf{M}[i, \cdot]$ denote the i'th row. The row support of $\mathbf{M}$ is the set of indices $I \subseteq \{1, \ldots, n\}$ such that $\mathbf{M}[i, \cdot] \neq \mathbf{0}$.

We say that a function ϵ is negligible in n if for every positive polynomial p there exists a constant c such that $\epsilon(n) < \frac{1}{p(n)}$ when $n > c$. Two ensembles $X = \{X_{\kappa,z}\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}$ and $Y = \{Y_{\kappa,z}\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}$ of binary random variables are said to be *statistically indistinguishable*, denoted by $X \approx_s Y$, if for all z it holds that $|\Pr[\mathcal{D}(X_{\kappa,z}) = 1] - \Pr[\mathcal{D}(Y_{\kappa,z}) = 1]|$ is negligible in κ for every probabilistic algorithm (distinguisher) $\mathcal{D}$. In case this only holds for computationally bounded (non-uniform probabilistic polynomial-time (PPT)) distinguishers we say that X and Y are *computationally indistinguishable* and denote it by $\approx_c$.

2.1 Coding Theory

For a vector $\boldsymbol{x} \in \mathbb{F}^n$, we denote the Hamming-weight of $\boldsymbol{x}$ by $\|\boldsymbol{x}\|_0 = |\{i \in [n] : \boldsymbol{x}[i] \neq 0\}|$. Let $\mathsf{C} \subset \mathbb{F}^n$ be a linear subspace of $\mathbb{F}^n$. We say that C is an $\mathbb{F}$-linear $[n, k, d]$ code, if C has dimension k and it holds for every nonzero $\boldsymbol{x} \in \mathsf{C}$ that $\|\boldsymbol{x}\|_0 \geq d$, *i.e.*, the minimum distance of C, denoted $\mathsf{dist}(\mathsf{C})$, is at least d. The distance $\mathsf{dist}(\mathsf{C}, \boldsymbol{x})$ between C and a vector $\boldsymbol{x} \in \mathbb{F}^n$ is the minimum of $\|\boldsymbol{c} - \boldsymbol{x}\|_0$ when $\boldsymbol{c} \in \mathsf{C}$. The rate of an $\mathbb{F}$-linear $[n, k, d]$ code is $\frac{k}{n}$ and its relative minimum distance is $\frac{d}{n}$.

A matrix $\mathbf{G} \in \mathbb{F}^{n \times k}$ is a generator matrix of C if $\mathsf{C} = \{\mathbf{G}\boldsymbol{x} : \boldsymbol{x} \in \mathbb{F}^k\}$, and we write $\mathsf{C}(\boldsymbol{x}) = \mathbf{G}\boldsymbol{x}$. The code C is systematic if it has a generator matrix $\mathbf{G}$ such that the submatrix given by the top k rows of $\mathbf{G}$ is the identity matrix $\mathbf{I} \in \mathbb{F}^{k \times k}$.

For an $\mathbb{F}$-linear $[n, k, d]$ code C, we denote by $\mathsf{C}^{\odot m}$ the *m-interleaved product* of C, which is defined by $\mathsf{C}^{\odot m} = \{\mathbf{C} \in \mathbb{F}^{n \times m} : \forall i \in [m] : \mathbf{C}[\cdot, i] \in \mathsf{C}\}$. In other

words, $\mathsf{C}^{\odot m}$ consists of all $\mathbb{F}^{n\times m}$ matrices for which all columns are in C. We can think of $\mathsf{C}^{\odot m}$ as a linear code with symbol alphabet $\mathbb{F}^m$, where we obtain codewords by taking m arbitrary codewords of C and bundling together the components of these codewords into symbols from $\mathbb{F}^m$. For a matrix $\mathbf{E} \in \mathbb{F}^{n\times m}$, $\|\mathbf{E}\|_0$ is the number of nonzero rows of $\mathbf{E}$, and the code $\mathsf{C}^{\odot m}$ has minimum distance at least d' if all nonzero $\mathbf{C} \in \mathsf{C}^{\odot m}$ satisfy $\|\mathbf{C}\|_0 \geq d'$. With this definition, it is easy to see that $\mathsf{dist}(\mathsf{C}^{\odot m}) = \mathsf{dist}(\mathsf{C})$

For an $\mathbb{F}$-linear $[n, k, d]$ code C, we denote by C^{*2} the *Schur square* of C, which is defined as the linear subspace of $\mathbb{F}^n$ generated by all the possible vectors of the form $\boldsymbol{v} * \boldsymbol{w}$ with $\boldsymbol{v}, \boldsymbol{w} \in \mathsf{C}$. This is an $[n, \hat{k}, \hat{d}]$ code where $\hat{k} \geq k$ and $\hat{d} \leq d$.

2.2 Interactive Proximity Testing and Linear Time Building Blocks

We will use the interactive proximity testing technique and corresponding linear time building blocks introduced in [16]. As stated in [16], this technique consists in the following argument: suppose we sample a function $\mathbf{H}$ from an almost universal family of linear hash functions (from $\mathbb{F}^m$ to $\mathbb{F}^\ell$), and we apply this to each of the rows of a matrix $\mathbf{X} \in \mathbb{F}^{n\times m}$, obtaining another matrix $\mathbf{X}' \in \mathbb{F}^{n\times \ell}$; because of linearity, if $\mathbf{X}$ belonged to an interleaved code $\mathsf{C}^{\odot m}$, then $\mathbf{X}'$ belongs to the interleaved code $\mathsf{C}^{\odot \ell}$. Theorem 1 states that we can test whether $\mathbf{X}$ is close to $\mathsf{C}^{\odot m}$ by testing instead if $\mathbf{X}'$ is close to $\mathsf{C}^{\odot \ell}$ (with high probability over the choice of the hash function) and moreover, if these elements are close to the respective codes, the set of rows that have to be modified in each of the matrices in order to correct them to codewords are the same.

Definition 1 (Almost Universal Linear Hashing [16]). *We say that a family $\mathcal{H}$ of linear functions $\mathbb{F}^n \to \mathbb{F}^s$ is ϵ-almost universal, if it holds for every non-zero $\mathbf{x} \in \mathbb{F}^n$ that*

$$\Pr_{\mathbf{H} \xleftarrow{\$} \mathcal{H}} [\mathbf{H}(\mathbf{x}) = 0] \leq \epsilon,$$

where $\mathbf{H}$ is chosen uniformly at random from the family $\mathcal{H}$. We say that $\mathcal{H}$ is universal, if it is $|\mathbb{F}^{-s}|$-almost universal. We will identify functions $H \in \mathcal{H}$ with their transformation matrix and write $\mathbf{H}(\mathbf{x}) = \mathbf{H} \cdot \mathbf{x}$.

Theorem 1 (Theorem 1 in [16]). *Let $\mathcal{H} : \mathbb{F}^m \to \mathbb{F}^{2s+t}$ be a family of $|\mathbb{F}|^{-2s}$-almost universal $\mathbb{F}$-linear hash functions. Further let C be an $\mathbb{F}$-linear $[n, k, s]$ code. Then for every $\mathbf{X} \in \mathbb{F}^{n\times m}$ at least one of the following statements holds, except with probability $|\mathbb{F}|^{-s}$ over the choice of $\mathbf{H} \xleftarrow{\$} \mathcal{H}$:*

1. *$\mathbf{X}\mathbf{H}^\top$ has distance at least s from $\mathsf{C}^{\odot(2s+t)}$*
2. *For every $\mathbf{C}' \in \mathsf{C}^{\odot(2s+t)}$ there exists a $\mathbf{C} \in \mathsf{C}^{\odot m}$ such that $\mathbf{X}\mathbf{H}^\top - \mathbf{C}'$ and $\mathbf{X} - \mathbf{C}$ have the same row support*

Remark 1 ([16]). If the first item in the statement of the Theorem does not hold, the second one must hold. Then we can efficiently recover a codeword $\mathbf{C}$ with distance at most $s - 1$ from $\mathbf{X}$ using erasure correction, given a codeword $\mathbf{C}' \in \mathsf{C}^{\odot(2s+t)}$ with distance at most $s - 1$ from $\mathbf{X}\mathbf{H}^\top$. More specifically, we

compute the row support of $\mathbf{X}\mathbf{H}^\top - \mathbf{C}'$, erase the corresponding rows of $\mathbf{X}$ and recover $\mathbf{C}$ from $\mathbf{X}$ using erasure correction[10]. The last step is possible as the distance between $\mathbf{X}$ and $\mathbf{C}$ is at most $s-1$.

In order to achieve linear time and optimal rate (*i.e.*, rate-1) in our constructions, we will need to instantiate interactive proximity testing with a family of linear time almost universal linear hash functions and a linear time encodable error correcting code that achieves rate 1. Theorems 3 and 6 from [16] guarantee that explicit constructions of such building blocks exist. The following theorem is a strengthening of Theorem 3 of [16] in that the output of the hash functions is guaranteed to be uniformly random given that its first l inputs are uniformly random. The full proof is given in the full version of this paper [18].

Theorem 2. *Fix a finite field $\mathbb{F}$ of constant size, let $s \in \mathbb{N}$ be a statistical security parameter, let $n \in \mathbb{N}$ and let $l = s + O(\log(n))$. Then there exists an explicit family $\mathcal{H} : \mathbb{F}^{l+n} \to \mathbb{F}^l$ of $|\mathbb{F}|^{-s}$-universal hash functions that can be represented by $O(s^2)$ bits and computed in time $O(n)$. Moreover, it holds for any function $H \in \mathcal{H}$ that if $\mathbf{x} = (x_1, \ldots, x_l, \ldots x_{l+n})$ is such that the $x_1, \ldots, x_l$ are independently uniform and $x_{l+1}, \ldots, x_{l+n}$ are independent of $x_1, \ldots, x_l$, then $H(\mathbf{x})$ is distributed uniformly random.*

2.3 Universal Composability

The protocols presented in this paper are proven secure in the Universal Composability (UC) framework introduced by Canetti in [12]. We refer the reader to the full version of this paper [18] and [12] for further details.

Adversarial Model: Our protocols will be proven secure against static and active adversaries: the adversary may deviate from the protocol in any arbitrary way but only corrupt parties before the protocol execution starts.

Setup Assumption: Since UC commitment protocols cannot be obtained in the plain model [13], they need a setup assumption, *i.e.*, a resource available to all parties before the protocol starts. In this work, our goal is to prove security in the $\mathcal{F}_{\text{COM}}$-hybrid model [12,14], where the parties have access to an ideal (non-homomorphic) commitment functionality (our constructions are described in the $\mathcal{F}_{\text{COM}}$-hybrid model for the sake of clarity, but they actually only need the underlying commitments to be extractable). Functionality $\mathcal{F}_{\text{COM}}$ is described in Figure 1. Notice that we describe a version of $\mathcal{F}_{\text{COM}}$ that operates with a set V of multiple receivers instead of a single receiver. However, $\mathcal{F}_{\text{COM}}$ can operate as a standard two-party commitment functionality with a single receiver by setting $V = \{V_1\}$, in which case it can be realized in the CRS model under different assumptions with security against static malicious adversaries by a number of protocols such as [13,27,8].

[10] Recall that erasure correction for linear codes can be performed efficiently via gaussian elimination.

A recent result by Camenisch *et al.* [11] shows that the "canonical" random oracle commitment realizes this functionality in the Global Random Oracle model without extra computational assumptions achieving security against static malicious adversaries. We observe that the protocol in [11] supports multiple receivers. In this protocol, the sender commits to a message $\boldsymbol{m}$ with randomness $\boldsymbol{r}$ by sending to the receiver the output $\boldsymbol{c}$ of the global random oracle when queried on $(\boldsymbol{r}, \boldsymbol{m})$ and opens by revealing $(\boldsymbol{r}, \boldsymbol{m})$, which allows the receiver to verify by querying the global random oracle with the pair $(\boldsymbol{r}, \boldsymbol{m})$ received as opening and checking that the response is equal to $\boldsymbol{c}$. Given that the random oracle functionality in this model is global, any number of receivers who have received the commitment and the opening can trivially obtain the same result in the verification.

Ideal Functionalities: In Section 3, we construct an additively homomorphic string commitment protocol that UC-realizes functionality $\mathcal{F}_{\text{AHCOM}}$, described in Figure 2. Similarly to a functionality of [16], $\mathcal{F}_{\text{AHCOM}}$ augments the standard multiple commitments functionality $\mathcal{F}_{\text{MCOM}}$ from [14] by introducing a command for adding two previously stored commitments and an abort command in the Commit Phase. Moreover, $\mathcal{F}_{\text{AHCOM}}$ gives an honest sender commitments to random messages instead of letting it submit a message as input, which can be straightforwardly used to commit to arbitrary messages with additive homomorphism as shown in [16]. In order to model corruptions, functionality $\mathcal{F}_{\text{AHCOM}}$ lets a corrupted sender choose the messages it wants to commit to. The abort is necessary to deal with inconsistent commitments that could be sent by a corrupted party. However, differently from [16] or [14], this functionality can operate with a set V of multiple receivers but only allows for a single opening of a batch of commitments, after which it halts, not allowing further commitments, additions or openings. Notice that this functionality can operate as a two-party commitment functionality with a single receiver by setting $V = \{V_1\}$. Section 4 shows how to modify the construction of Section 3 to obtain a protocol that UC-realizes the augmented functionality $\mathcal{F}_{\text{MHCOM}}$ (Figure 3), which also allows for multiplication of committed values.

Functionality $\mathcal{F}_{\text{COM}}$

$\mathcal{F}_{\text{COM}}$ is parameterized by commitment length λ. $\mathcal{F}_{\text{COM}}$ interacts with a sender P, a set of receivers $V = \{V_1, \ldots, V_t\}$ and an adversary $\mathcal{S}$ and proceeds as follows:

- **Commit Phase**: Upon receiving a message $(\mathsf{commit}, sid, ssid, P, V, \boldsymbol{m})$ from P where $\boldsymbol{m} \in \{0,1\}^{\lambda}$, record the tuple $(ssid, P, V, \boldsymbol{m})$ and send $(\mathsf{receipt}, sid, ssid, P, V)$ to every receiver $V_i \in V$ and $\mathcal{S}$. Ignore subsequent commit messages with the same $ssid$.
- **Open Phase**: Upon receiving a message $(\mathsf{reveal}, sid, ssid)$ from P, if a tuple $(ssid, P, V, \boldsymbol{m})$ was previously recorded, then send $(\mathsf{reveal}, sid, ssid, P, V, \boldsymbol{m})$ to every receiver $V_i \in V$ and $\mathcal{S}$. Otherwise, ignore.

Fig. 1. Functionality $\mathcal{F}_{\text{COM}}$.

Functionality $\mathcal{F}_{\text{AHCOM}}$

$\mathcal{F}_{\text{AHCOM}}$ interacts with a sender P, a set of receivers $V = \{V_1, \ldots, V_t\}$ and an adversary $\mathcal{S}$ and proceeds as follows:

- **Commit Phase**: The length of the committed messages λ is fixed and known to all parties.
  - If P is honest, upon receiving a message $(\mathsf{commit}, sid, ssid, P, V)$ from P, sample a random $\boldsymbol{m} \leftarrow \{0,1\}^\lambda$, record the tuple $(ssid, P, V, \boldsymbol{m})$, send the message $(\mathsf{commit}, sid, ssid, P, V, \boldsymbol{m})$ to P and send the message $(\mathsf{receipt}, sid, ssid, P, V)$ to every receiver $V_i \in V$ and $\mathcal{S}$. Ignore any future commit messages with the same $ssid$ from P to V.
  - If P is corrupted, upon receiving a message $(\mathsf{commit}, sid, ssid, P, V, \boldsymbol{m})$ from P, where $\boldsymbol{m} \in \{0,1\}^\lambda$, record the tuple $(ssid, P, V, \boldsymbol{m})$ and send the message $(\mathsf{receipt}, sid, ssid, P, V)$ to every receiver $V_i \in V$ and $\mathcal{S}$. Ignore any future commit messages with the same $ssid$ from P to V.
  - If a message $(\mathsf{abort}, sid, ssid)$ is received from $\mathcal{S}$, the functionality halts.
- **Addition**: Upon receiving a message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P, V)$ from P: If tuples $(ssid_1, P, V, \boldsymbol{m}_1)$, $(ssid_2, P, V, \boldsymbol{m}_2)$ were previously recorded and $ssid_3$ is unused, record $(ssid_3, P, V, \boldsymbol{m}_1 + \boldsymbol{m}_2)$ and send the message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P, V, \mathsf{success})$ to P, every receiver $V_i \in V$ and $\mathcal{S}$.
- **Open Phase**: Upon receiving a message $(\mathsf{reveal}, sid, ssid_1, \ldots, ssid_o)$ from P, for every $ssid \in \{ssid_1, \ldots, ssid_o\}$, if a tuple $(ssid, P, V, \boldsymbol{m})$ was previously recorded, then send $(\mathsf{reveal}, sid, ssid, P, V, \boldsymbol{m})$ to every receiver $V_i \in V$ and $\mathcal{S}$, if not, send nothing. Finally, halt.

Fig. 2. Functionality $\mathcal{F}_{\text{AHCOM}}$

Functionality $\mathcal{F}_{\text{MHCOM}}$

Augment the functionality $\mathcal{F}_{\text{AHCOM}}$ (Figure 2) with the step:

- **Multiplication**: Upon receiving a message $(\mathsf{mult}, sid, ssid_1, ssid_2, ssid_3, P, V)$ from P: If tuples $(ssid_1, P, V, \boldsymbol{m}_1)$, $(ssid_2, P, V, \boldsymbol{m}_2)$ were previously recorded and $ssid_3$ is unused, record $(ssid_3, P, V, \boldsymbol{m}_1 * \boldsymbol{m}_2)$ and send the message $(\mathsf{mult}, sid, ssid_1, ssid_2, ssid_3, P, V, \mathsf{success})$ to P, every receiver $V_i \in V$ and $\mathcal{S}$.

Fig. 3. Functionality $\mathcal{F}_{\text{MHCOM}}$

3 Rate-1 Linear Time Additively Homomorphic Commitments

In this section, we construct a linear time additively homomorphic commitment protocol that achieves amortized rate-1 and linear time in the length of committed messages assuming an extractable (not homomorphic) commitment and a PRG as building blocks. Protocol Π_{AHCOM} realizes $\mathcal{F}_{\text{AHCOM}}$, which only allows for commitments to random messages. Interestingly, in this case we can achieve sublinear communication complexity in the commitment phase while maintaining rate-1 in the opening phase. Even though committing to random messages is useful for a number of applications (*e.g.* [23]) that $\mathcal{F}_{\text{AHCOM}}$ is sufficient for building a protocol Π_{ARBHCOM} that commits to arbitrary messages achieving rate-1

and running in linear time as discussed in [16]. Essentially, Protocol Π_{AHCOM} achieves the same asymptotic efficiency as the former best UC commitment scheme [16], while supporting multiple verifiers and without requiring OT in the preprocessing phase, resulting in better concrete efficiency.

The main idea is to use a "delayed watchlist" mechanism where the sender first commits to seeds that will be stretched by a PRG to instantiate the watchlist but only allows the receivers to learn the watch bits in a later point, at which the receivers choose a random subset of the seed commitments to be opened. Basically, the watchlist is viewed as a matrix $\mathbf{R} = \mathbf{R}_0 + \mathbf{R}_1$ such that, for each row of $\mathbf{R}$, the receiver learns only a row from either $\mathbf{R}_0$ or $\mathbf{R}_1$ without revealing to the sender which one. Instead of using a number of 1-out-of-2 random OTs to obtain seeds that are stretched to generate each line of $\mathbf{R}_0$ or $\mathbf{R}_1$ in the beginning of the protocol as in previous works, the receiver relies on simple commitments to each seed sent by the sender. This scheme achieves rate-1 using similar techniques as [16]: first having the sender adjust the bottom bits of the watchlist matrix $\mathbf{R}$ so that its columns are codewords of random strings (in the top bits of $\mathbf{R}$) and then using interactive proximity testing to convince the receiver that these columns are indeed "very close" to codewords. In order to "open" a commitment, the sender reveals the columns from both $\mathbf{R}_0$ and $\mathbf{R}_1$ corresponding to that commitment, allowing a receiver who knows rows from each of these matrices to check that the revealed column vector corresponds to the watchlist with high probability. However, in our new scheme, the receiver only chooses which commitments to seeds will be revealed after the sender has sent this opening information. Otherwise, the sender would learn which rows of $\mathbf{R}_0$ or $\mathbf{R}_1$ the receiver would check, being able to open commitments to arbitrary messages. Protocol Π_{AHCOM} is described in Figures 4 and 5.

In comparison to the protocol of [16], our scheme realizes a functionality with a caveat that only one opening of a batch of commitments is allowed (after which it terminates). However, this limited functionality is sufficient for a number of applications that we discuss in later sections. Moreover, our protocol has two important properties that the scheme of [16] lacks: it is public coin and supports multiple receivers. Notice that the watch bits of the receiver (represented by a row from either $\mathbf{R}_0$ or $\mathbf{R}_1$) are chosen at random but in public by the receiver. Hence, given an underlying commitment that support multiple receivers (*e.g.*, the canonical random oracle commitment scheme), it is sufficient to have the receivers run a simple commit-then-open coin tossing protocol to choose the watch bits they will learn, then have the sender publicly open his seed commitments. Interestingly, having the receivers broadcast their coin tossing commitments at the beginning of the protocol (before the sender broadcasts opening information), allows the simulator to both equivocate and extract commitments solely by extracting the underlying commitments. Notice that the simulator can equivocate a commitment by knowing in advance the watch bits to be learned by the receivers and extract a commitment by learning the whole watchlist, which are fixed in the sender's seed commitments. In order to eliminate interaction with

the receivers, the random watch bits to be opened can be selected with the help of a random oracle following the Fiat-Shamir heuristic.

Efficiency. We achieve the same asymptotic complexity as [16] but with a pre-processing phase that can be instantiated with lower concrete complexity since it only requires extractable commitments. All phases of the Π_{AHCOM} run in linear time (requiring a constant number of operations per committed bit) when we use a linear time PRG (*i.e.*, with a constant number of operations per generated bit [30]) a linear time encodable code C (*e.g.* the one from [16]) and a linear time linear almost universal hash function $\mathbf{H}$ (*e.g.* the one from [16]). The cost of the calls to $\mathcal{F}_{\text{AHCOM}}$ is amortized over the number of commitments, which does not need to be very large if $\mathcal{F}_{\text{AHCOM}}$ is instantiated with cheap random oracle based commitments. The commitment phase achieves sublinear communication complexity when committing to random messages, since a rate-1 $[n, k, s]$-code C is used and only $\mathbf{W}, \mathbf{T_0}, \mathbf{T_1}$ (of size $O(1)$) are exchanged. Even if the trick from [16] is used to commit to arbitrary messages, only k extra bits need to be sent per message. In this case, our protocol achieves rate-1, meaning that the amortized overhead per committed bit is $o(1)$ for a sufficiently large number of commitments. The opening phase as described in Figure 5 does not achieve rate-1, since the sender has to send both $\mathbf{A_0}[\cdot, j]$ $\mathbf{A_1}[\cdot, j]$. However, it can be modified to achieve rate-1 using the same technique from [16], where a batch of commitments are opened by performing interactive proximity testing on a matrix $\mathbf{A}'$ containing the columns of $\mathbf{A}$ corresponding to the commitments to be opened. The receivers can use another coin-tossing to select a hash function $\mathbf{H}$, then the sender sends $\mathbf{A}'$, $\mathbf{T_0}' = \mathbf{A_0}'\mathbf{H}$ and $\mathbf{T_1}' = \mathbf{A_1}'\mathbf{H}$. The receivers check that $\mathbf{A}'\mathbf{H} = \mathbf{T_0}' + \mathbf{T_1}'$, that all columns in $\mathbf{A}'$ are in C and that $\mathbf{\Delta T_0}' + (\mathbf{I} - \mathbf{\Delta})\mathbf{T_1}' = \mathbf{B}'\mathbf{H}$, where $\mathbf{B}'$ contains the columns from $\mathbf{B}$ corresponding to the commitments being checked. This technique can be proven secure with the same techniques used for the case of a corrupt sender.

Security Analysis. For the sake of clarity, we will prove Protocol Π_{AHCOM}'s security in the $\mathcal{F}_{\text{COM}}$-hybrid model, *i.e.* assuming access to an ideal functionality for commitments. The proof of security for Protocol Π_{AHCOM} is very similar to that of the scheme of [16], with the exception that all information the simulator needs to extract and equivocate commitments will be obtained from $\mathcal{F}_{\text{COM}}$ instead of an OT functionality. However, our simulator will only rely on the fact that it can extract the messages sent by the adversary to $\mathcal{F}_{\text{COM}}$ before it opens its commitments. Essentially, our simulators only need an underlying commitment scheme that is extractable, not a full blown UC commitment scheme (which would also allow the simulator to open the underlying commitments to arbitrary messages). The security of Protocol Π_{AHCOM} is formally stated in Theorem 3.

Theorem 3. *Protocol Π_{AHCOM} UC-realizes $\mathcal{F}_{\text{AHCOM}}$ in the $\mathcal{F}_{\text{COM}}$-hybrid model with computational security against a static adversary. Formally, there exists a simulator $\mathcal{S}$ such that for every static adversary $\mathcal{A}$, and any environment $\mathcal{Z}$, the environment cannot distinguish Π_{AHCOM} composed with $\mathcal{F}_{\text{COM}}$ and $\mathcal{A}$ from $\mathcal{S}$ composed with $\mathcal{F}_{\text{AHCOM}}$. That is,* $\mathsf{IDEAL}_{\mathcal{F}_{\text{AHCOM}},\mathcal{S},\mathcal{Z}} \approx_c \mathsf{HYBRID}^{\mathcal{F}_{\text{COM}}}_{\Pi_{\text{AHCOM}},\mathcal{A},Z}$.

Protocol Π_{AHCOM}

Let C be a systematic binary linear $[n, k, s]$ code, where s is the statistical security parameter and n is $k + O(s)$. Let $\mathcal{H}$ be a family of linear almost universal hash functions $\mathbf{H} : \{0,1\}^m \to \{0,1\}^l$. Let $\mathsf{PRG} : \{0,1\}^\ell \to \{0,1\}^{m+l}$ be a pseudorandom generator. Protocol Π_{AHCOM} is run by a sender P and a set of receivers $V = \{V_1, \ldots, V_t\}$, who interact with $\mathcal{F}_{\text{COM}}$ and proceed as follows:

Commitment Phase

1. On input $(\mathsf{commit}, sid, ssid_1, \ldots, ssid_m, P, V)$, P proceeds as follows:
   (a) For $i \in [n]$ and $j \in \{0,1\}$, sample $\boldsymbol{s}_{i,j} \xleftarrow{\$} \{0,1\}^\ell$ and send $(\mathsf{commit}, sid, ssid_{i,j}, P, V, \boldsymbol{s}_{i,j})$ to $\mathcal{F}_{\text{COM}}$.
   (b) Compute $\mathbf{R_j}[i, \cdot] = \mathsf{PRG}(\boldsymbol{s}_{i,j})$ and set $\mathbf{R} = \mathbf{R_0} + \mathbf{R_1}$ so that $\mathbf{R_0}, \mathbf{R_1}$ forms an additive secret sharing of $\mathbf{R}$.
   (c) Adjust the bottom $n - k$ rows of $\mathbf{R}$ so that all columns are codewords in C by constructing a matrix $\mathbf{W}$ with dimensions as $\mathbf{R}$ and 0s in the top k rows, such that $\mathbf{A} := \mathbf{R} + \mathbf{W} \in \mathsf{C}^{\odot m+l}$ (recall that C is systematic). Set $\mathbf{A_0} = \mathbf{R_0}, \mathbf{A_1} = \mathbf{R_1} + \mathbf{W}$ and broadcast $(sid, ssid_1, \ldots, ssid_m, \mathbf{W})$ (only sending the bottom $n - k = O(s)$ rows).
2. Upon receiving all messages $(\mathsf{receipt}, sid, ssid_{i,j}, P, V)$ from $\mathcal{F}_{\text{COM}}$ and $(sid, ssid_1, \ldots, ssid_m, \mathbf{W})$ from P, every receiver $V_i \in V$ proceeds as follows:
   (a) Sample $\boldsymbol{r}_i \xleftarrow{\$} \{0,1\}^n$ and $\boldsymbol{r}'_i \xleftarrow{\$} \{0,1\}^\ell$, and send $(\mathsf{commit}, sid, ssid, V_i, V', \boldsymbol{r_i})$ and $(\mathsf{commit}, sid, ssid', V_i, V', \boldsymbol{r_i}')$ to $\mathcal{F}_{\text{COM}}$[a], where $V' = P \cup V \setminus V_i$.
   (b) Upon receiving $(\mathsf{receipt}, sid, ssid, V_j, V')$ and $(\mathsf{receipt}, sid, ssid', V_j, V')$ from $\mathcal{F}_{\text{COM}}$ for all $V_j \in V \setminus V_i$, send $(\mathsf{reveal}, sid, ssid')$ to $\mathcal{F}_{\text{COM}}$.
   (c) Upon receiving $(\mathsf{reveal}, sid, ssid', V_j, V', \boldsymbol{r_j}')$ from $\mathcal{F}_{\text{COM}}$ for all $V_j \in V \setminus V_i$, set $\boldsymbol{r}' = \boldsymbol{r_1}' \oplus \ldots \oplus \boldsymbol{r_t}'$.
3. Upon receiving $(\mathsf{commit}, sid, ssid, V_i, V')$ and $(\mathsf{reveal}, sid, ssid', V_j, V', \boldsymbol{r_j}')$ from $\mathcal{F}_{\text{COM}}$ for all $V_j \in V$, P proceeds as follows:
   (a) Use $\boldsymbol{r}' = \boldsymbol{r_1}' \oplus \ldots \oplus \boldsymbol{r_t}'$ as a seed for a random function $\mathbf{H} \in \mathcal{H}$ (note that we identify the function with its matrix and all functions in $\mathcal{H}$ are linear).
   (b) Set matrices $\mathbf{P}$, $\mathbf{P_0}$ and $\mathbf{P_1}$ as the first l columns of $\mathbf{A}$, $\mathbf{A_0}$ and $\mathbf{A_1}$, respectively, and remove these columns from $\mathbf{A}$, $\mathbf{A_0}$ and $\mathbf{A_1}$. Renumber the remaining columns of $\mathbf{A}$, $\mathbf{A_0}$ and $\mathbf{A_1}$ from 1 and associate each $ssid_i$ (commitment id from step 1) with a different column index in these matrices. Notice that $\mathbf{P} = \mathbf{P_0} + \mathbf{P_1}$.
   (c) For $i \in \{0,1\}$, compute $\mathbf{T}_i = \mathbf{A}_i\mathbf{H} + \mathbf{P}_i$ and broadcast $(sid, ssid_1, \ldots, ssid_m, \mathbf{T_0}, \mathbf{T_1})$. Note that $\mathbf{AH} + \mathbf{P} = \mathbf{A_0H} + \mathbf{P_0} + \mathbf{A_1H} + \mathbf{P_1} = \mathbf{T_0} + \mathbf{T_1}$, and $\mathbf{AH} + \mathbf{P} \in \mathsf{C}^{\odot l}$.

[a] We abuse notation and assume that each receiver V_i in Π_{AHCOM} has access to an instance of $\mathcal{F}_{\text{COM}}$ that takes as message with the appropriate length where it acts as sender and where all other receivers plus sender P act as receivers.

Fig. 4. Commit phase for the protocol Π_{AHCOM}.

Proof. We give the proof in the full version of this paper [18]. Note that constructing a simulator for the case where all parties are honest is trivial. Hence, the theorem follows by establishing security against an adversary that corrupts

Protocol Π_{AHCOM}

Addition of Commitments

1. On input $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P, V)$, P finds indexes i and j corresponding to $ssid_1$ and $ssid_2$ respectively and check that $ssid_3$ is unused. P appends the column $\mathbf{A}[\cdot, i] + \mathbf{A}[\cdot, j]$ to $\mathbf{A}$, likewise appends to $\mathbf{A_0}$ and $\mathbf{A_1}$ the sum of their i-th and j-th columns, and associates $ssid_3$ with the new column index. P broadcasts $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$. Note that this maintains the properties $\mathbf{A} = \mathbf{A_0} + \mathbf{A_1}$ and $\mathbf{A} \in \mathsf{C}^{\odot m'}$, where m' is the current number of columns (after appending columns for addition results).
2. Upon receiving $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$, every receiver $V_i \in V$ stores the message.

Opening

1. On input $(\mathsf{reveal}, sid, ssid_1, \ldots, ssid_o)$, P finds the set $J = \{j_1, \ldots, j_o\}$ of indexes associated to $ssid_1, \ldots, ssid_o$ and broadcasts $(sid, ssid_1, \ldots, ssid_o, (\mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j])_{j \in J})$.
2. Upon receiving message $(sid, ssid_1, \ldots, ssid_o, (\mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j])_{j \in J})$, every $V_i \in V$ sends $(\mathsf{reveal}, sid, ssid)$ to $\mathcal{F}_{\text{COM}}$ and waits for $(\mathsf{reveal}, sid, ssid, V_j, V', \boldsymbol{r_j})$ from $\mathcal{F}_{\text{COM}}$ for all $V_j \in V \setminus V_i$. V_i sets $\boldsymbol{r} = \boldsymbol{r_1} \oplus \cdots \oplus \boldsymbol{r_t}$ and sets the diagonal matrix $\boldsymbol{\Delta}$ such that it contains $\boldsymbol{r}[1], \ldots, \boldsymbol{r}[n]$ in the diagonal.
3. Upon receiving $(\mathsf{reveal}, sid, ssid, V_j, V', \boldsymbol{r_j})$ from $\mathcal{F}_{\text{COM}}$ for all $V_j \in V$, P sets $\boldsymbol{r} = \boldsymbol{r_1} \oplus \ldots \oplus \boldsymbol{r_t}$, sends $(\mathsf{reveal}, sid, ssid_{i,\boldsymbol{r}[i]})$ to $\mathcal{F}_{\text{COM}}$ for $i \in [n]$ and halts.
4. Upon receiving $(\mathsf{reveal}, sid, ssid_{i,\boldsymbol{r}[i]}, P, V, \boldsymbol{s}_{i,\boldsymbol{r}[i]})$ from $\mathcal{F}_{\text{COM}}$ for $i \in [n]$, every receiver $V_j \in V$ proceeds as follows:
   (a) Compute $\mathbf{S}[i, \cdot] = \mathsf{PRG}(\boldsymbol{s}_{i,\boldsymbol{r}[i]})$, obtaining a matrix $\mathbf{S}$. Note that each row of $\mathbf{S}$ is a row from either $\mathbf{R_0}$ or $\mathbf{R_1}$, which form an additive secret sharing of $\mathbf{R}$ held by P. Set $\mathbf{B} = \boldsymbol{\Delta}\mathbf{W} + \mathbf{S}$. Define the matrix $\mathbf{Q}$ as the first l columns of $\mathbf{B}$ and remove these columns from $\mathbf{B}$, renumbering the remaining columns from 1. Note that, for $\mathbf{A}$ from the commitment phase, $\mathbf{A} = \mathbf{A_0} + \mathbf{A_1}$, $\mathbf{B} = \boldsymbol{\Delta}\mathbf{A_1} + (\mathbf{I} - \boldsymbol{\Delta})\mathbf{A_0}$, $\mathbf{A} \in \mathsf{C}^{\odot m}$, *i.e.*, $\mathbf{A}$ initially held by P is additively shared and for each row index, V knows either a row from $\mathbf{A_0}$ or from $\mathbf{A_1}$.
   (b) Check that $\boldsymbol{\Delta}\mathbf{T_1} + (\mathbf{I} - \boldsymbol{\Delta})\mathbf{T_0} = \mathbf{BH} + \mathbf{Q}$ and that $\mathbf{T_0} + \mathbf{T_1} \in \mathsf{C}^{\odot l}$. If any check fails, abort. Notice that $\mathbf{T_0}, \mathbf{T_1}$ form an additive sharing of $\mathbf{AH} + \mathbf{P}$, where V knows some of the shares, namely the rows of $\mathbf{BH} + \mathbf{Q}$.
   (c) For every message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$ received from P, append $\mathbf{B}[\cdot, j] + \mathbf{B}[\cdot, i]$ to $\mathbf{B}$, where i and j are the index corresponding to $ssid_1$ and $ssid_2$ respectively and associate $ssid_3$ with the new column index. Note that this maintains the property $\mathbf{B} = \boldsymbol{\Delta}\mathbf{A_1} + (\mathbf{I} - \boldsymbol{\Delta})\mathbf{A_0}$.
   (d) For every $j \in J$, check that $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j] \in \mathsf{C}$ and that, for $i \in [n]$, it holds that $\mathbf{B}[i, j] = \mathbf{A}_{\boldsymbol{r}[i]}[i, j]$ (recall that $\boldsymbol{r}[i]$ is the i-th entry on the diagonal of $\boldsymbol{\Delta}$). If all checks succeed, for every $j \in J$, output the first k positions in $\mathbf{A_0}[\cdot, j] + \mathbf{A_1}[\cdot, j]$ as the opened string and halt. Otherwise, abort by outputting $(sid, ssid_j, \perp)$.

Fig. 5. Addition of commitments and opening phase for the protocol Π_{AHCOM}.

P and all but one receiver in V or an adversary who corrupts all receivers in V. See the full version for each of these.

4 Achieving Multiplicative Homomorphism

In this section, we modify our additively homomorphic commitment protocol described Section 3 (protocol Π_{AHCOM}) so that it is also homomorphic for (coordinatewise) multiplication of messages. That is, if we denote the scheme from Section 3 by $\mathtt{com}$, our goal is that given commitments $\mathtt{com}(\boldsymbol{a})$, $\mathtt{com}(\boldsymbol{b})$ the prover can construct a commitment $\mathtt{com}(\boldsymbol{a} * \boldsymbol{b})$. In order to do this we need to introduce a second auxiliary commitment scheme $\mathtt{prodcom}$, also described below.

Both $\mathtt{com}$ and $\mathtt{prodcom}$ can be obtained by changing the instantiation of two of the building blocks of protocol Π_{AHCOM}. Namely, at the core of the construction of the commitment scheme in Section 3 (as well as in the ones from [16,17,22]) there is a linear error correcting code C, which is used to encode the message and which needs to have a large enough minimum distance; and there is the 2-out-of-2 additive secret sharing scheme Add_2, which is applied to each coordinate of the encoding. Our modifications are as follows: first, we need a linear code C such that also its *(Schur) square* C^{*2} has a large enough minimum distance. We will use C as the linear code in $\mathtt{com}$ and C^{*2} as the linear code in $\mathtt{prodcom}$ (with a certain caveat described below). As for the secret sharing schemes, we will use the *replicated secret sharing scheme* RSS_3 (described below) for $\mathtt{com}$ and the additive 3-out-of-3 Add_3 secret sharing scheme for $\mathtt{prodcom}$. RSS_3 is the secret sharing scheme where the secret $s \in \{0,1\}$ is additively split into three parts, i.e., $s = r_0 + r_1 + r_2$ where r_0, r_1 are uniformly random and independent, and the shares are defined to be the pairs $s_0 = (r_0, r_1)$, $s_1 = (r_1, r_2)$, $s_2 = (r_2, r_0)$. RSS_3 is a multiplicative secret sharing scheme, which means that shares of s, s' can locally be transformed into shares by Add_3 of the product $s \cdot s'$. More precisely, $s \cdot s' = t_0 + t_1 + t_2$, where $t_i = r_i r'_i + r_i r'_{i+1} + r'_i r_{i+1}$ (where sums in the indices are modulo 3) and note that all this information is contained in the i-th shares s_i, s'_i of s and s'. The rationale for the choices of codes and secret sharing schemes is then that from the watchlists of $\mathtt{com}(\boldsymbol{a}), \mathtt{com}(\boldsymbol{b})$ a verifier can compute a watchlist to a commitment $\mathtt{prodcom}(\boldsymbol{a} * \boldsymbol{b})$. Indeed, given the j-th share (in RSS_3) of the i-th coordinates $(\mathsf{C}(\boldsymbol{a}))_i, (\mathsf{C}(\boldsymbol{b}))_i$ the verifier can determine the j-th share (in Add_3) of $(\mathsf{C}(\boldsymbol{a}) * \mathsf{C}(\boldsymbol{b}))_i$, and note $\mathsf{C}(\boldsymbol{a}) * \mathsf{C}(\boldsymbol{b})$ is a codeword in C^{*2} having $\boldsymbol{a} * \boldsymbol{b}$ as the vector of its first k coordinates.

But our goal is to construct $\mathtt{com}(\boldsymbol{a} * \boldsymbol{b})$ rather than $\mathtt{prodcom}(\boldsymbol{a} * \boldsymbol{b})$. We do that as follows: the prover constructs commitments $\mathtt{com}(\boldsymbol{y})$, $\mathtt{prodcom}(\boldsymbol{y})$ of a random vector $\boldsymbol{y}$ with both commitment schemes, where for every coordinate i, the verifier will later request to open the share with the same index r_i in $\mathtt{prodcom}(\boldsymbol{y})$ as he does for $\mathtt{com}(\boldsymbol{a}), \mathtt{com}(\boldsymbol{b}), \mathtt{com}(\boldsymbol{y})$ (note that for $\mathtt{com}$ that means the additive shares indexed by r_i and $r_i + 1$). The sender needs to prove that $\mathtt{com}(\boldsymbol{y})$, $\mathtt{prodcom}(\boldsymbol{y})$ are indeed commitments to the same vector, which will be detailed later. From $\mathtt{com}(\boldsymbol{a}), \mathtt{com}(\boldsymbol{b})$ the prover constructs all the shares in $\mathtt{prodcom}(\boldsymbol{a} * \boldsymbol{b})$ as mentioned above, and then announces all three additive shares

of $\boldsymbol{a} * \boldsymbol{b} - \boldsymbol{y}$. For each coordinate i, the receiver can determine the r_i-th share of this vector from the watchlists of $\mathtt{com}(\boldsymbol{a}), \mathtt{com}(\boldsymbol{b}), \mathtt{prodcom}(\boldsymbol{y})$ and contrast this with the information that the prover opens. Now assuming the verifier does not abort, the prover and verifier can simply construct $\mathtt{com}(\boldsymbol{a} * \boldsymbol{b})$ by adding $\boldsymbol{a} * \boldsymbol{b} - \boldsymbol{y}$ to $\mathtt{com}(\boldsymbol{y})$. [11] We need to address some technical details: commitments with $\mathtt{prodcom}$ are to messages of length k' (the dimension of C^{*2}) rather than messages of length k and in general it can happen that $k' > k$, so when we say $\mathtt{prodcom}(\boldsymbol{y})$ we mean that the commitment is to a vector $\boldsymbol{y}||\boldsymbol{z}$ where $\boldsymbol{z}$ is of length $k'-k$. Moreover, initially we cannot choose the random vectors we commit to since these are generated pseudorandomly from the seeds, so the prover will need to send some correction information in order to commit to the same value in the two schemes. In order to do that, and simultaneously prepare to prove that $\mathtt{com}(\boldsymbol{y})$ and $\mathtt{prodcom}(\boldsymbol{y})$ are commitments to the same vector $\boldsymbol{y}$, we define the linear code $\widetilde{\mathsf{C}}$ defined as the concatenation of C and C^{*2}. More precisely,

$$\widetilde{\mathsf{C}} = \{(\boldsymbol{y}, \boldsymbol{c}, \boldsymbol{y}, \boldsymbol{c'}) : (\boldsymbol{y}, \boldsymbol{c}) \in \mathsf{C}, (\boldsymbol{y}, \boldsymbol{c'}) \in \mathsf{C}^{*2}\}. \tag{1}$$

The prover, having used the PRGs to construct pairs of random vectors $\boldsymbol{r}, \boldsymbol{r'}$ in $\{0,1\}^n$ and additive splittings of them, will concatenate the two vectors and send correction information $\boldsymbol{z} \in \{0,1\}^{2n}$ so that $(\boldsymbol{r}||\boldsymbol{r'}) - \boldsymbol{z} \in \widetilde{\mathsf{C}}$ (as before, the first k bits of $\boldsymbol{z}$ can be taken to be 0, so the prover needs to send only $2n-k$ bits). Now given a batch of supposed codewords of this form the interactive proximity testing technique is applied so that the sender proves they are indeed codewords in $\widetilde{\mathsf{C}}$, and therefore they are associated to commitments $(\mathtt{com}(\boldsymbol{y}), \mathtt{prodcom}(\boldsymbol{y}))$. Note that since the first n coordinates of the codewords in $\widetilde{\mathsf{C}}$ are codewords in C, this test also guarantees all properties of the interactive proximity test for the additive case, so we do not need to perform that one separately.
We note that $\widetilde{d} = \mathsf{dist}(\widetilde{\mathsf{C}}) \geq \mathsf{dist}(\mathsf{C}^{*2})$, [12] so we need a lower bound on $\mathsf{dist}(\mathsf{C}^{*2})$ to obtain the same guarantees as in the additive case. Furthermore, a difference with the proof for the additive-only commitment scheme is that now the verifier sees 2 out of 3 additive shares of the first n coordinates and 1 out of 3 coordinates of the last n, which affects the cheating probabilities of a corrupt prover: we will show that it is enough to assume that $\mathsf{dist}(\mathsf{C}^{*2}) > \beta s$, where $\beta = 1/(\log_2 3 - 1) = 1.709...$ (which satisfies $(2/3)^\beta = 1/2$), in order to guarantee that the cheating prover can succeed with probability at most 2^{-s}. Protocol Π_{MHCOM} is described in Figures 6, 7 and 8. Notice that for consistency with the notation of Section 3, we describe our fully homomorphic commitment protocol for random messages. However, a commitment to chosen messages $\boldsymbol{m}$ can be created using the protocol Π_{MHCOM} simply sending $\boldsymbol{c} = \boldsymbol{m} - \boldsymbol{a}$, where $\boldsymbol{a} = \pi_{[k]}(\mathbf{A}[\cdot, i])$ is one of the random messages that the prover gets in the commit phase of Π_{MHCOM}. Now, in order

[11] More precisely, the last share of each coordinate of $\mathsf{C}(\boldsymbol{y})$ is added with the corresponding (now public) coordinate of $\mathsf{C}(\boldsymbol{a} * \boldsymbol{b} - \boldsymbol{y})$.

[12] One may think that the tighter lower bound $\mathsf{dist}(\widetilde{\mathsf{C}}) \geq \mathsf{dist}(\mathsf{C}) + \mathsf{dist}(\mathsf{C}^{*2})$ holds, but this is not necessarily true if the dimension k' of C^{*2} is larger than k, as in that case there will be codewords of the form $(\boldsymbol{0^k}, \boldsymbol{0^{n-k}}, \boldsymbol{0^k}, \boldsymbol{c'})$ where $\boldsymbol{c'} \neq \boldsymbol{0^{n-k}}$. Indeed take $(\boldsymbol{0^k}, \boldsymbol{c'})$ to be the encoding by C^{*2} of $(\boldsymbol{0^k}||\boldsymbol{z})$ for a nonzero $\boldsymbol{z} \in \{0,1\}^{k'-k}$

to allow multiplication of commitments to chosen messages it is enough that all the players locally adjust the shares of the random messages used as OTP keys (*e.g.*, the prover P adds $\mathsf{C}(\boldsymbol{c})$ to $\mathbf{A}_2[\cdot, i]$ and every receiver in V adds $\boldsymbol{\Delta}\mathsf{C}(\boldsymbol{c})$ and $\boldsymbol{\Delta}'\mathsf{C}(\boldsymbol{c})$ to $\mathbf{B}[\cdot, i]$ and $\mathbf{B}'[\cdot, i]$, respectively) and then execute the multiplication step as detailed in Figure 7.

Finally notice that for the sake of simplicity, in the commit phase of Protocol Π_{MHCOM} we use the same notation and the same construction both for random messages that are actually input to commitments (or used to construct a commitment to a chosen message as explained above) and for the auxiliary random messages that are needed in the multiplication step (*i.e.*, $\boldsymbol{y}$ in the notation used in the introduction of this section), so that all those messages are encoded in columns of the big matrix $\tilde{\mathbf{A}}$. However, committing with `prodcom`, and hence creating and manipulating the last n rows of the matrix $\tilde{\mathbf{A}}$ (what we call $\hat{\mathbf{A}}$), is only necessary for the random messages used in the multiplication step, and could be saved for the remaining random messages. On the other hand, the current structure of the commit phase, where we do not distinguish between the two roles for the random messages, allows us to use only a single interactive proximity test instead of two (*i.e.*, one for C as in protocol Π_{AHCOM} to guarantee the additive property and another one for $\widetilde{\mathsf{C}}$ and the auxiliary random messages to guarantee that the same value $\boldsymbol{y}$ is encoded using C and C^{*2}).

Security Analysis. The proof of security for Protocol Π_{MHCOM} is similar to that of Π_{AHCOM}. Indeed, the following Theorem 4 can be proved by adapting the description of the simulators for the security proof of the additive-only construction to the new watchlist setting (*i.e.*, three additive shares instead of two, of which the verifier knows either two - in the base commitment given by matrix $\mathbf{A}$ - or one - in the product commitment given by $\widehat{\mathbf{A}}$) and adding to both simulators the step to simulate the multiplication command. More details are given in the full version [18].

Theorem 4. *Protocol Π_{MHCOM} UC realizes $\mathcal{F}_{\text{MHCOM}}$ in the $\mathcal{F}_{\text{COM}}$-hybrid model with computational security against a static adversary. Formally, there exists a simulator $\mathcal{S}$ such that for every static adversary $\mathcal{A}$, and any environment $\mathcal{Z}$, the environment cannot distinguish Π_{MHCOM} composed with $\mathcal{F}_{\text{COM}}$ and $\mathcal{A}$ from $\mathcal{S}$ composed with $\mathcal{F}_{\text{MHCOM}}$. That is,* $\mathsf{IDEAL}_{\mathcal{F}_{\text{MHCOM}},\mathcal{S},\mathcal{Z}} \approx_c \mathsf{HYBRID}^{\mathcal{F}_{\text{COM}}}_{\Pi_{\text{MHCOM}},\mathcal{A},Z}$.

Efficiency. Since we commit to every random message with both `com` and `prodcom`, the total length of the commitment will be $2n - k + o(k)$ bits per message of k bits. For chosen messages we need to add an extra k bits per message for a total of $2n$ bits. If C has rate R, our commitments have then rate $R/2$. Moreover, for multiplying two commitments the prover needs to have created an additional commitment of a random message with both `com` and `prodcom` (hence communicating $2n$ bits), and then communicate all shares of a related commitment with `prodcom` (the $\boldsymbol{w}_i$'s in the protocol), which amounts to $3n$ bits. So the communication of this step is $5n$ bits. The question is then what rates we can have under our new requirements on $\mathsf{dist}(\mathsf{C}^{*2})$.

Families of binary codes $\{\mathsf{C}_n\}$ with constant rate (of C_n) and constant relative minimum distance of C_n^{*2} exist based on algebraic geometry [28]. For fixed values of the security parameter s the families of cyclic codes constructed in [15] give better rates. As an example, for $s = 60$, where our protocol needs $\mathsf{dist}(\mathsf{C}^{*2}) \geq 103$, Table 2 in [15] gives a $[4095, 338]$ cyclic code with $\mathsf{dist}(\mathsf{C}^{*2}) \geq 135$, which has rate around 0.08. Hence the commitments will have rate 0.04.[13] We need to send $25k$ bits per k-bit message we commit to, and $62.5k$ bits to construct a commitment to the product of two messages.

5 Applications to Efficient Zero-Knowledge Arguments

In this section, we outline how to use a variant of the homomorphic commitments constructed in Section 3 and 4 to compile a certain class of public coin interactive proof system into public coin honest-verifier zero-knowledge proof systems. Using the Fiat-Shamir heuristic, we can convert such a zero-knowledge proof system into a non-interactive zero-knowledge proof system. As an application, we can improve a recent construction of zkSNARKs [31] in a certain parameter regime. Specifically, the zkSNARK construction of [31] uses additively homomorphic vector commitments[14] to transform a public coin interactive proof system into a zero-knowledge protocol. The commitments in [31] are instantiated using number-theoretic assumptions. One of the core ideas of [31] is that general algebraic relations between commitments can be reduced to linear relations between vector-commitments in a way that only induces a constant additional overhead for low-degree relations. The construction of [31] is general enough that it can be instantiated with homomorphic commitment schemes with some additional properties. We remark though that [31] utilizes an additional optimization which relies on *compressing* homomorphic commitments, which is not available in our setting.

Our main observation is that for this application the unveil of the commitments in the protocol of [31] can be delayed until the very end of the protocol, which makes this protocol compatible with our commitment scheme.

The notion of interactive proof system we focus on will be resettably sound public coin interactive proofs with *algebraic verifier*. Such a proof system proceeds in t rounds, where in each round i the prover sends a message p_i, upon which the verifier answers with a uniformly random message v_i. We require all the messages p_i and v_i to be vectors over a field $\mathbb{F}$. After the conversation is over, the verifier evaluates a system of low degree polynomials $F_1, \ldots, F_s$ in the p_i and v_i and accepts if all F_i evaluate to 0, otherwise it rejects. At the heart of this kind of protocol is the sum-check protocol, which lets a prover prove statements of the form $\sum_{x \in \{0,1\}^n} P(x) = L$, where $P \in \mathbb{F}[X_1, \ldots, X_n]$ is a low-degree polynomial and $L \in \mathbb{F}$.

[13] And naturally from this one can also obtain a $[4095 \cdot \ell, 338 \cdot \ell]$-code with the same minimum distance of its square, by simply applying the $[4095, 338]$ to each block of 338 bits of the message.

[14] In [31] they are referred to as *multi-commitments*

Protocol Π_{MHCOM}

Let C be a systematic binary linear $[n,k]$ code, such that C^{*2} is also systematic and satisfies $\mathsf{dist}(\mathsf{C}^{*2}) \geq \beta s$, where $\beta = 1/(\log_2 3 - 1)$ and s is the statistical security parameter. Let $\widetilde{\mathsf{C}}$ be the code defined in (1). Let $\mathcal{H}$ be a family of linear almost universal hash functions $\mathbf{H} : \{0,1\}^m \to \{0,1\}^l$. Let $\mathsf{PRG} : \{0,1\}^\ell \to \{0,1\}^{m+l}$ be a pseudorandom generator. Protocol Π_{MHCOM} is run by a sender P and a set of receivers $V = \{V_1, \ldots, V_t\}$, who interact with $\mathcal{F}_{\text{COM}}$ as follows:

Commitment Phase

1. On input $(\mathsf{commit}, sid, ssid_1, \ldots, ssid_m, P, V)$, P proceeds as follows:
   (a) For $i \in [n]$ and $j \in \{0,1,2\}$, sample $\boldsymbol{s}_{i,j} \xleftarrow{\$} \{0,1\}^\ell$, $\widehat{\boldsymbol{s}}_{i,j} \xleftarrow{\$} \{0,1\}^\ell$ and send $(\mathsf{commit}, sid, ssid_{i,j}, P, V, \boldsymbol{s}_{i,j})$, $(\mathsf{commit}, sid, \widehat{ssid}_{i,j}, P, V, \widehat{\boldsymbol{s}}_{i,j})$ to $\mathcal{F}_{\text{COM}}$.
   (b) Compute $\mathbf{R_j}[i,\cdot] = \mathsf{PRG}(\boldsymbol{s}_{i,j})$ and $\widehat{\mathbf{R}}_\mathbf{j}[i,\cdot] = \mathsf{PRG}(\widehat{\boldsymbol{s}}_{i,j})$ and set $\mathbf{R} = \mathbf{R_0} + \mathbf{R_1} + \mathbf{R_2}$ and $\widehat{\mathbf{R}} = \widehat{\mathbf{R}}_\mathbf{0} + \widehat{\mathbf{R}}_\mathbf{1} + \widehat{\mathbf{R}}_\mathbf{2}$.
   (c) Adjust the bottom $n-k$ rows of $\mathbf{R}$ so that all columns are codewords in C by constructing a matrix $\mathbf{W}$ with dimensions as $\mathbf{R}$ and 0s in the top k rows, such that $\mathbf{A} := \mathbf{R} + \mathbf{W} \in \mathsf{C}^{\odot m+l}$ (recall that C is systematic). Set $\mathbf{A_0} = \mathbf{R_0}, \mathbf{A_1} = \mathbf{R_1}, \mathbf{A_2} = \mathbf{R_2} + \mathbf{W}$.
   (d) Adjust $\widehat{\mathbf{R}}$ so that all columns are codewords in C^{*2} and the first k rows are the same as in $\mathbf{A}$ by constructing a matrix $\widehat{\mathbf{W}}$ with dimensions as $\widehat{\mathbf{R}}$ such that $\widehat{\mathbf{A}} := \widehat{\mathbf{R}} + \widehat{\mathbf{W}} \in (\mathsf{C}^{*2})^{\odot m+l}$ and $\widehat{\mathbf{A}}[i,\cdot] = \mathbf{A}[i,\cdot]$ for all $i \in [k]$. Set $\widehat{\mathbf{A}}_\mathbf{0} = \widehat{\mathbf{R}}_\mathbf{0}, \widehat{\mathbf{A}}_\mathbf{1} = \widehat{\mathbf{R}}_\mathbf{1}, \widehat{\mathbf{A}}_\mathbf{2} = \widehat{\mathbf{R}}_\mathbf{2} + \widehat{\mathbf{W}}$ and broadcast $(sid, ssid_1, \ldots, ssid_m, \mathbf{W}, \widehat{\mathbf{W}})$ (sending the bottom $n-k$ rows of $\mathbf{W}$ and the entire matrix $\widehat{\mathbf{W}}$).
2. Upon receiving all $(\mathsf{receipt}, sid, ssid_{i,j}, P, V)$ from $\mathcal{F}_{\text{COM}}$ and $(sid, ssid_1, \ldots, ssid_m, \mathbf{W}, \widehat{\mathbf{W}})$ from P, every $V_i \in V$ proceeds as follows:
   (a) Sample $\boldsymbol{r_i} \xleftarrow{\$} \mathbb{Z}_3^n$, $\boldsymbol{r_i}' \xleftarrow{\$} \{0,1\}^\ell$ and send $(\mathsf{commit}, sid, ssid, V_i, V', \boldsymbol{r_i})$ and $(\mathsf{commit}, sid, ssid', V_i, V', \boldsymbol{r_i}')$ to $\mathcal{F}_{\text{COM}}$, where $V' = P \cup V \setminus V_i$.
   (b) and (c) as is the commit phase of Π_{AHCOM} (Figure 4).
3. Upon receiving $(\mathsf{commit}, sid, ssid, V_i, V')$ and $(\mathsf{reveal}, sid, ssid', V_j, V', \boldsymbol{r_j}')$ from $\mathcal{F}_{\text{COM}}$ for all $V_j \in V$, P proceeds as follows:
   (a) Use $\boldsymbol{r}' = \boldsymbol{r_1}' \oplus \ldots \oplus \boldsymbol{r_t}'$ as a seed for a random function $\mathbf{H} \in \mathcal{H}$.
   (b) Define the matrices $\widetilde{\mathbf{A}} = \begin{pmatrix} \mathbf{A} \\ \widehat{\mathbf{A}} \end{pmatrix}$ and $\widetilde{\mathbf{A}}_\mathbf{i} = \begin{pmatrix} \mathbf{A_i} \\ \widehat{\mathbf{A}}_\mathbf{i} \end{pmatrix}$ for $i \in \{0,1,2\}$. Note that $\widetilde{\mathbf{A}} \in \widetilde{\mathsf{C}}^{\odot m+l}$ and $\widetilde{\mathbf{A}} = \widetilde{\mathbf{A}}_\mathbf{0} + \widetilde{\mathbf{A}}_\mathbf{1} + \widetilde{\mathbf{A}}_\mathbf{2}$. Set the matrices $\widetilde{\mathbf{P}}$ and $\widetilde{\mathbf{P}}_i$ as the first l columns of $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{A}}_i$, respectively, and remove these columns from $\widetilde{\mathbf{A}}$, $\widetilde{\mathbf{A}}_i$, $\mathbf{A}$, $\mathbf{A}_i$, $\widehat{\mathbf{A}}$, $\widehat{\mathbf{A}}_i$ for $i \in \{0,1,2\}$. Renumber the remaining columns from 1 and associate each commitment $ssid_i$ (commitment id from step 1) with a different column in these matrices. Notice that $\widetilde{\mathbf{P}} = \widetilde{\mathbf{P}}_\mathbf{0} + \widetilde{\mathbf{P}}_\mathbf{1} + \widetilde{\mathbf{P}}_\mathbf{2}$.
   (c) For $i \in \{0,1,2\}$, compute the matrix $\widetilde{\mathbf{T}}_i = \widetilde{\mathbf{A}}_i\mathbf{H} + \widetilde{\mathbf{P}}_i$ and broadcast $(sid, ssid_1, \ldots, ssid_m, \widetilde{\mathbf{T}}_\mathbf{0}, \widetilde{\mathbf{T}}_\mathbf{1}, \widetilde{\mathbf{T}}_\mathbf{2})$. Note that $\widetilde{\mathbf{A}}\mathbf{H} + \widetilde{\mathbf{P}} = \widetilde{\mathbf{T}}_\mathbf{0} + \widetilde{\mathbf{T}}_\mathbf{1} + \widetilde{\mathbf{T}}_\mathbf{2}$, and $\widetilde{\mathbf{A}}\mathbf{H} + \widetilde{\mathbf{P}} \in \widetilde{\mathsf{C}}^{\odot l}$.

Fig. 6. Commit phase for the protocol Π_{MHCOM}.

While it can be shown that any constant round proof system can be immediately compiled into a non-interactive argument system via the Fiat-Shamir

Protocol Π_{MHCOM}

Addition of Commitments

1. On input $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P, V)$, P finds indexes i and j corresponding to $ssid_1$ and $ssid_2$ respectively and check that $ssid_3$ is unused. P appends the column $\mathbf{A}[\cdot, i] + \mathbf{A}[\cdot, j]$ to $\mathbf{A}$, likewise appends to $\mathbf{A_0}$, $\mathbf{A_1}$, $\mathbf{A_2}$ the sum of their i-th and j-th columns, and associates $ssid_3$ with the new column index. P broadcasts $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$ to V.
2. Upon receiving $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$, every $V_i \in V$ stores the message.

Multiplication of Commitments

1. On input $(\mathsf{mult}, sid, ssid_1, ssid_2, ssid_3, P, V)$, P finds indexes i and j corresponding to $ssid_1$ and $ssid_2$ respectively and check that $ssid_3$ is unused. Then, P proceeds as follows:
   (a) For $l \in \{0, 1, 2\}$, compute $\boldsymbol{v}_l = \mathbf{A}_l[\cdot, i] * \mathbf{A}_l[\cdot, j] + \mathbf{A}_l[\cdot, i] * \mathbf{A}_{l+1}[\cdot, j] + \mathbf{A}_{l+1}[\cdot, i] * \mathbf{A}_l[\cdot, j]$. Note that $\boldsymbol{v_0}, \boldsymbol{v_1}, \boldsymbol{v_2}$ are shares of $\mathbf{A}[\cdot, i] * \mathbf{A}[\cdot, j]$ in the scheme Add_3 and known to P only. Let h be the index of the first unused column from $\mathbf{A}$ and $\widehat{\mathbf{A}}$, compute $\boldsymbol{w}_l = \boldsymbol{v}_l - \widehat{\mathbf{A}}_l[\cdot, h]$ for $l = 0, 1, 2$ and broadcast $(sid, ssid, h, \boldsymbol{w_0}, \boldsymbol{w_1}, \boldsymbol{w_2})$ to V. Note that $\boldsymbol{w_0}, \boldsymbol{w_1}, \boldsymbol{w_2}$ are shares of $\mathbf{A}[\cdot, i] * \mathbf{A}[\cdot, j] - \widehat{\mathbf{A}}[\cdot, h]$ in the scheme Add_3 and are known to $P \cup V$.
   (b) Let $\boldsymbol{u} = \pi_{[k]}(\boldsymbol{w_0} + \boldsymbol{w_1} + \boldsymbol{w_2})$ (*i.e.*, $\boldsymbol{u}$ consists of the first k components of $\mathbf{A}[\cdot, i] * \mathbf{A}[\cdot, j] - \widehat{\mathbf{A}}[\cdot, h]$), append the columns $\mathbf{A}[\cdot, h] + \mathsf{C}(\boldsymbol{u})$ and $\mathbf{A_2}[\cdot, h] + \mathsf{C}(\boldsymbol{u})$ to $\mathbf{A}$ and $\mathbf{A_2}$, respectively. Append the column $\mathbf{A}_i[\cdot, h]$ to $\mathbf{A}_i$ for $i = 0, 1$ and associate $ssid_3$ with the new column index. Note that since $\pi_{[k]}(\widehat{\mathbf{A}}[\cdot, h]) + \pi_{[k]}(\mathbf{A}[\cdot, h])$, for $l \in \{1, \ldots, k\}$ the l-th component of the newly appended column in $\mathbf{A}$ is equal to $\mathbf{A}[l, i] * \mathbf{A}[l, j]$. Broadcast $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$ to V.
2. Upon receiving $(\mathsf{mult}, sid, ssid_1, ssid_2, ssid_3)$, every $V_i \in V$ stores the message.

Note that this maintains the properties $\mathbf{A} = \mathbf{A_0} + \mathbf{A_1} + \mathbf{A_2}$ and $\mathbf{A} \in \mathsf{C}^{\odot m'}$, where m' is the current number of columns.

Opening (Part 1)

1. On input $(\mathsf{reveal}, sid, ssid_1, \ldots, ssid_o)$, P finds the set $J = \{j_1, \ldots, j_o\}$ of indexes associated to $ssid_1, \ldots, ssid_o$ and broadcasts $(sid, ssid_1, \ldots, ssid_o, (\mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j], \mathbf{A_2}[\cdot, j])_{j \in J})$.
2. Upon receiving message $(sid, ssid_1, \ldots, ssid_o, (\mathbf{A_0}[\cdot, j], \mathbf{A_1}[\cdot, j], \mathbf{A_2}[\cdot, j])_{j \in J})$, every receiver $V_i \in V$ sends $(\mathsf{reveal}, sid, ssid)$ to $\mathcal{F}_{\text{COM}}$ and waits for $(\mathsf{reveal}, sid, ssid, V_j, V', \boldsymbol{r_j})$ from $\mathcal{F}_{\text{COM}}$ for all $V_j \in V \setminus V_i$. V_i sets $\boldsymbol{r} = \boldsymbol{r_1} + \cdots + \boldsymbol{r_t}$ (where the sum is in $\mathbb{Z}_3^n$) and sets the diagonal matrices $\boldsymbol{\Delta}, \boldsymbol{\Delta}'$ such that the i-th element in $\boldsymbol{\Delta}$ (resp. $\boldsymbol{\Delta}'$) is 1 if $\boldsymbol{r}[i] = 2$ (resp. $\boldsymbol{r}[i] = 1$) and 0 otherwise.
3. Upon receiving $(\mathsf{reveal}, sid, ssid, V_j, V', \boldsymbol{r_j})$ from $\mathcal{F}_{\text{COM}}$ for all $V_j \in V$, P sets $\boldsymbol{r} = \boldsymbol{r_1} + \ldots + \boldsymbol{r_t}$, sends $(\mathsf{reveal}, sid, ssid_{i, \boldsymbol{r}[i]})$, $(\mathsf{reveal}, sid, ssid_{i, \boldsymbol{r}[i]+1})$ and $(\mathsf{reveal}, sid, \widehat{ssid}_{i, \boldsymbol{r}[i]})$ to $\mathcal{F}_{\text{COM}}$ for $i = 1, \ldots, n$ and halts.

Fig. 7. Addition and multiplication steps, and opening phase for the protocol Π_{MHCOM}.

Protocol Π_{MHCOM}

Opening (Part 2)

4. Upon receiving the messages $(\mathsf{reveal}, sid, ssid_{i,\boldsymbol{r}[i]}, P, V, \boldsymbol{s}_{i,\boldsymbol{r}[i]})$, $(\mathsf{reveal}, sid, ssid_{i,\boldsymbol{r}[i]+1}, P, V, \boldsymbol{s}_{i,\boldsymbol{r}[i]+1})$ and $(\mathsf{reveal}, sid, \widehat{ssid}_{i,\boldsymbol{r}[i]}, P, V, \widehat{\boldsymbol{s}}_{i,\boldsymbol{r}[i]})$ from $\mathcal{F}_{\text{COM}}$ for $i \in \{1, \ldots, n\}$, every receiver $V_j \in V$ proceeds as follows:
   (a) Compute $\mathbf{S}[i,\cdot] = \mathsf{PRG}(\boldsymbol{s}_{i,\boldsymbol{r}[i]})$, $\mathbf{S}'[i,\cdot] = \mathsf{PRG}(\boldsymbol{s}_{i,\boldsymbol{r}[i]+1})$ and $\widehat{\mathbf{S}}[i,\cdot] = \pi_{\mu+l}\big(\mathsf{PRG}(\widehat{\boldsymbol{s}}_{i,\boldsymbol{r}[i]})\big)$ obtaining matrices $\mathbf{S}$, $\mathbf{S}'$ and $\widehat{\mathbf{S}}$. Note for each i, the i-th row of $\mathbf{S}$, $\mathbf{S}'$, $\widehat{\mathbf{S}}$ will equal the i-th row of $\mathbf{R}_{\boldsymbol{r}[\mathbf{i}]}$, $\mathbf{R}_{\boldsymbol{r}[\mathbf{i}]+\mathbf{1}}$, $\widehat{\mathbf{R}}_{\boldsymbol{r}[\mathbf{i}]}$ respectively. Set $\mathbf{B} = \boldsymbol{\Delta}\mathbf{W} + \mathbf{S}$, $\mathbf{B}' = \boldsymbol{\Delta}'\mathbf{W} + \mathbf{S}'$ and $\widehat{\mathbf{B}} = \boldsymbol{\Delta}\widehat{\mathbf{W}} + \widehat{\mathbf{S}}$. Define the matrices[a] $\mathbf{Q}$, $\mathbf{Q}'$, $\widehat{\mathbf{Q}}$ as the first l columns of $\mathbf{B}$, $\mathbf{B}'$, $\widehat{\mathbf{B}}$ and remove these columns from the latter matrices, renumbering the remaining columns from 1.
   (b) Notice that $\widetilde{\mathbf{T}}_{\mathbf{0}}, \widetilde{\mathbf{T}}_{\mathbf{1}}, \widetilde{\mathbf{T}}_{\mathbf{2}}$ form an additive sharing of $\widetilde{\mathbf{A}}\mathbf{H} + \widetilde{\mathbf{P}}$, and the verifiers know some of the shares, namely the rows of $\mathbf{B}\mathbf{H} + \mathbf{Q}$ and $\mathbf{B}'\mathbf{H} + \mathbf{Q}'$ (shares for the first n rows of $\widetilde{\mathbf{A}}\mathbf{H} + \widetilde{\mathbf{P}}$) and the rows of $\widehat{\mathbf{B}}\mathbf{H} + \widehat{\mathbf{Q}}$ (shares for the last n rows). For $i \in \{0, 1, 2\}$, parse $\widetilde{\mathbf{T}}_i$ as $\widetilde{\mathbf{T}}_i = \begin{pmatrix} \mathbf{T}_i \\ \widehat{\mathbf{T}}_i \end{pmatrix}$. Check that $\mathbf{B}\mathbf{H} + \mathbf{Q} = \boldsymbol{\Delta}\mathbf{T}_{\mathbf{2}} + \boldsymbol{\Delta}'\mathbf{T}_{\mathbf{1}} + (1 - \boldsymbol{\Delta} - \boldsymbol{\Delta}')\mathbf{T}_{\mathbf{0}}$, $\mathbf{B}'\mathbf{H} + \mathbf{Q}' = \boldsymbol{\Delta}\mathbf{T}_{\mathbf{0}} + \boldsymbol{\Delta}'\mathbf{T}_{\mathbf{2}} + (1 - \boldsymbol{\Delta} - \boldsymbol{\Delta}')\mathbf{T}_{\mathbf{1}}$ and $\widehat{\mathbf{B}}\mathbf{H} + \widehat{\mathbf{Q}} = \boldsymbol{\Delta}\widehat{\mathbf{T}}_{\mathbf{2}} + \boldsymbol{\Delta}'\widehat{\mathbf{T}}_{\mathbf{1}} + (1 - \boldsymbol{\Delta} - \boldsymbol{\Delta}')\widehat{\mathbf{T}}_{\mathbf{0}}$, and that $\mathbf{T}_{\mathbf{0}} + \mathbf{T}_{\mathbf{1}} + \mathbf{T}_{\mathbf{2}} \in \mathsf{C}^{\odot \ell}$. If any check fails, abort.
   (c) For every $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$ received from P, append $\mathbf{B}[\cdot, a] + \mathbf{B}[\cdot, b]$ to $\mathbf{B}$ and append $\mathbf{B}'[\cdot, a] + \mathbf{B}'[\cdot, b]$ to $\mathbf{B}'$ (a, b are the index corresponding to $ssid_1$, $ssid_2$ respectively and $ssid_3$ is associated with the new column index). For every $(\mathsf{mult}, sid, ssid_1, ssid_2, ssid_3)$ received from P:
      – given $(sid, ssid, h, \boldsymbol{w}_{\mathbf{0}}, \boldsymbol{w}_{\mathbf{1}}, \boldsymbol{w}_{\mathbf{2}})$, check that $\boldsymbol{w}_{\mathbf{0}} + \boldsymbol{w}_{\mathbf{1}} + \boldsymbol{w}_{\mathbf{2}} \in \mathsf{C}^{*2}$ and $\boldsymbol{w}_{\boldsymbol{r}[i]} = \mathbf{B}[\cdot, a] * \mathbf{B}[\cdot, b] + \mathbf{B}[\cdot, a] * \mathbf{B}'[\cdot, b] + \mathbf{B}'[\cdot, a] * \mathbf{B}[\cdot, b] + \widehat{\mathbf{B}}[\cdot, h]$;
      – let $\boldsymbol{u} = \pi_{[k]}(\boldsymbol{w}_{\mathbf{0}} + \boldsymbol{w}_{\mathbf{1}} + \boldsymbol{w}_{\mathbf{2}})$, append the columns $\mathbf{B}[\cdot, h] + \boldsymbol{\Delta}\mathsf{C}(\boldsymbol{u})$ and $\mathbf{B}'[\cdot, h] + \boldsymbol{\Delta}'\mathsf{C}(\boldsymbol{u})$ to $\mathbf{B}$ and $\mathbf{B}'$, respectively.
      Note that the properties detailed in footnote[a] are maintained.
   (d) For every $j \in J$, check that $\mathbf{A}_{\mathbf{0}}[\cdot, j] + \mathbf{A}_{\mathbf{1}}[\cdot, j] + \mathbf{A}_{\mathbf{2}}[\cdot, j] \in \mathsf{C}$ and that, for $i = 1, \ldots, n$, it holds that $\mathbf{B}[i, j] = \mathbf{A}_{\boldsymbol{r}[\mathbf{i}]}[i, j]$ and $\mathbf{B}'[i, j] = \mathbf{A}_{\boldsymbol{r}[\mathbf{i}]+\mathbf{1}}[i, j]$. If all checks succeed, for every $j \in J$, output the first k positions in $\mathbf{A}_{\mathbf{0}}[\cdot, j] + \mathbf{A}_{\mathbf{1}}[\cdot, j] + \mathbf{A}_{\mathbf{2}}[\cdot, j]$ as the opened string and halts. Otherwise, abort by outputting $(sid, ssid_j, \perp)$.

[a] Note that we have $\mathbf{A} = \mathbf{A}_{\mathbf{0}} + \mathbf{A}_{\mathbf{1}} + \mathbf{A}_{\mathbf{2}}$, $\mathbf{B} = \boldsymbol{\Delta}\mathbf{A}_{\mathbf{2}} + \boldsymbol{\Delta}'\mathbf{A}_{\mathbf{1}} + (1 - \boldsymbol{\Delta} - \boldsymbol{\Delta}')\mathbf{A}_{\mathbf{0}}$ and $\mathbf{B}' = \boldsymbol{\Delta}\mathbf{A}_{\mathbf{0}} + \boldsymbol{\Delta}'\mathbf{A}_{\mathbf{2}} + (1 - \boldsymbol{\Delta} - \boldsymbol{\Delta}')\mathbf{A}_{\mathbf{1}}$. This means that $\mathbf{A}$ held by P is shared in the replicated secret sharing scheme RSS_3 and for each row index, V knows one share (*i.e.*, V knows the corresponding rows from exactly two of the matrices $\mathbf{A}_{\mathbf{0}}$, $\mathbf{A}_{\mathbf{1}}$, $\mathbf{A}_{\mathbf{2}}$). Moreover, $\widehat{\mathbf{A}} = \widehat{\mathbf{A}}_{\mathbf{0}} + \widehat{\mathbf{A}}_{\mathbf{1}} + \widehat{\mathbf{A}}_{\mathbf{2}}$ and $\widehat{\mathbf{B}} = \boldsymbol{\Delta}\widehat{\mathbf{A}}_{\mathbf{2}} + \boldsymbol{\Delta}'\widehat{\mathbf{A}}_{\mathbf{1}} + (\mathbf{I} - \boldsymbol{\Delta} - \boldsymbol{\Delta}')\widehat{\mathbf{A}}_{\mathbf{0}}$ *i.e.*, $\widehat{\mathbf{A}}$ held by P is shared in the additive secret sharing scheme Add_3 and for each row index, V knows one share (V knows the corresponding row from exactly one of the matrices $\widehat{\mathbf{A}}_{\mathbf{0}}$, $\widehat{\mathbf{A}}_{\mathbf{1}}$, $\widehat{\mathbf{A}}_{\mathbf{2}}$).

Fig. 8. Opening phase (continued) for the protocol Π_{MHCOM}.

heuristic [20], super-constant round protocols need to fulfil a stronger soundness

property called *resettable soundness* for the Fiat-Shamir transform to result in a sound protocol.

We will now outline how to compile any resettable sound public coin interactive proof system into an honest-verifier zero-knowledge proof systems in a way that only slightly increases the communication complexity and only affects the efficiency of prover and verifier by a small constant factor.

The basic idea of the transformation is simple and follows the paradigm of *committed conversations* [5]. The prover and verifier run the interactive proof system with the modification that instead of sending its messages in the plain, the prover sends commitments to its messages. After the protocol is over the prover convinces the verifier that the commitment values pass the verification equations $F_1, \ldots, F_s$. The homomorphic property of the commitments will be used to implement this check efficiently. While our protocol Π_{MHCOM} does support evaluation of low degree polynomials, we will focus on linear/affine verification equations $F_1, \ldots, F_s$ and will therefore rely on the additively homomorphic commitment scheme Π_{AHCOM}, with several modifications which are discussed in the full version [18].

Instantiation We will now discuss instantiating the hyrax protocol of [31] with the modified version of the commitment scheme Π_{AHCOM}.

To prove satisfiability of an algebraic circuit of depth d, width G and input/witness size $|w|$, the hyrax protocol has proof size $(10d\log(G) + \sqrt{|w|}) \cdot \kappa$ assuming that a group element in a DLOG-hard group $\mathbb{G}$ has size κ. The verifier runtime is $O(\sqrt{|w|} + d \cdot \log(G))$ whereas the prover runtime is linear in the size of the circuit C.

Replacing the DLOG-based homomorphic commitment in the hyrax protocol with our commitment protocol Π_{AHCOM} as outlined above, the main optimization which is not available is compression of the witness w. Consequently, in our instantiation proof size will depend linearly on the size of the witness $|w|$.

One of the key ideas in the hyrax protocol is to reduce all algebraic relations between commitments to linear relations between vector commitments, an idea also used in bulletproofs [10]. In this way, general algebraic relations can be proven using a protocol which just supports linear relations between vectors. This transformation only incurs a small constant factor additional overhead. Omitting details, there are three main steps. In the first step reduce multiplicative relations to linear relations, in the second step show that many linear relations can be compressed into a single linear relation, and in. the third step step reduce linear relations between commitments to linear relations between vector commitments. All three steps are implemented using a Schnorr-style protocol. In [31] these transformations are provided for the concrete case of DLOG-based commitments, but these ideas can be implemented using arbitrary homomorphic vector commitments.

The main improvement of our protocol over [31] is that we only rely on simple private key primitives. On the turn side, our vector-commitments are not compressing, which leads to the proof-size to depend linearly on the witness-size $|w|$ instead of $\sqrt{|w|}$. However, the proof size does not depend multiplicatively on

the computational security parameter κ, but rather on $|\mathbb{F}|$, which is a statistical security parameter an can therefore be chosen much smaller. Consequently, we get an advantage in terms of proof-size whenever the proof-size is dominated by d rather than $|w|$.

6 Applications to Secure Multiparty Computation

6.1 Committed MPC

A recent work by Frederiksen *et al.* [21] has shown that additively homomorphic commitments can be leveraged to construct efficient preprocessed MPC. However, their "Committed MPC" protocol requires a multiparty commitment functionality that allows for multiple senders and for computing linear combinations between commitments generated by different senders. We will show a generic construction of such a protocol from functionality $\mathcal{F}_{\text{AHCOM}}$ that can be instantiated with Protocol Π_{AHCOM}, achieving significantly better efficiency than the construction of [21].

Functionality $\mathcal{F}_{\text{MSAHCOM}}$. Our protocol will realize the multiparty additively homomorphic commitment functionality from [21] with the difference that it will only allow for a single batch verification of opened commitments. While it allows for openings before verification, the validity of those will not be ensured by $\mathcal{F}_{\text{MSAHCOM}}$, which will let the adversary choose any value to be provided as an opening. $\mathcal{F}_{\text{MSAHCOM}}$ will allow for a single verification phase where all parties check whether the openings they have received are valid, after which the functionality halts. This functionality is sufficient for realizing the "Committed MPC" protocol of [21], since the parties can use the intermediate (non-verified) openings to compute the protocol and in the end verify that the result is correct. Other small differences is that we omit the Partial Open interface used to open a commitment to a single receiver and provide an interface for single addition operations. Notice that our procedures for opening a commitment for all receivers can be trivially adapted to opening towards a specific receiver by sending the corresponding messages only to that receiver and that single additions of commitments can be trivially used for computing linear combinations as in the functionality of [21]. We present Functionality $\mathcal{F}_{\text{MSAHCOM}}$ in Figure 9.

Protocol Π_{MSAHCOM}. While a generic construction of such a protocol from any two-party additively homomorphic commitment scheme is presented in [21], we can significantly simplify and improve the efficiency of this construction departing from a multi-receiver scheme as defined in $\mathcal{F}_{\text{AHCOM}}$. We construct a protocol where every party acts both as sender and receiver of all commitments. In this protocol, each party first uses $\mathcal{F}_{\text{AHCOM}}$ to commit to random values towards the others. A joint random commitment in the new multi-sender protocol is defined as the commitment to the sum of all random messages contained in the individual commitments by each party. Linear combinations between joint commitments can be computed by having each party (acting as a sender in the

Functionality $\mathcal{F}_{\text{MSAHCOM}}$

$\mathcal{F}_{\text{MSAHCOM}}$ is parameterized by $n \in \mathbb{N}$. $\mathcal{F}_{\text{MSAHCOM}}$ interacts with a set of parties $P = \{P_1, \ldots, P_t\}$ and an adversary $\mathcal{S}$ (who may abort at any time):

- **Init** Upon receiving (init, sid) from all parties in P, forward the message to $\mathcal{S}$ and initialize empty lists raw and actual.
- **Commit:** Upon receiving $(\mathsf{commit}, sid, \mathcal{I})$ from all parties in P where $\mathcal{I}$ is a set of unused identifiers, for every $ssid \in \mathcal{I}$, sample a random $\boldsymbol{x}_{ssid} \xleftarrow{\$} \mathbb{F}^k$, set $\mathsf{raw}[ssid] = \boldsymbol{x}_{ssid}$ and send $(\mathsf{commit} - \mathsf{recorded}, sid, \mathcal{I})$ to all parties P and $\mathcal{S}$.
- **Input:** Upon receiving $(\mathsf{input}, sid, ssid, P_i, \boldsymbol{y})$ from $P_i \in P$ and $(\mathsf{input}, sid, ssid, P_i)$ from all other parties in P, if $\mathsf{raw}[ssid] = \boldsymbol{x}_{ssid} \neq \perp$, set $\mathsf{raw}[ssid] = \perp$, set $\mathsf{actual}[ssid] = \boldsymbol{y}$ and send $(\mathsf{input} - \mathsf{recorded}, sid, ssid, P_i)$ to all parties in P and $\mathcal{S}$.
- **Random:** Upon receiving $(\mathsf{random}, sid, ssid)$ from all parties in P, if $\mathsf{raw}[ssid] = \boldsymbol{x}_{ssid} \neq \perp$, set $\mathsf{actual}[ssid] = \boldsymbol{x}_{ssid}$, set $\mathsf{raw}[ssid] = \perp$ and send $(\mathsf{random} - \mathsf{recorded}, sid, ssid)$ to all parties P and $\mathcal{S}$.
- **Addition**: Upon receiving a message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$ from all parties in P: if $\mathsf{actual}[ssid] = \boldsymbol{x}_{ssid} \neq \perp$ for $ssid \in \{ssid_1, ssid_2\}$ and $\mathsf{raw}[ssid_3] = \mathsf{actual}[ssid_3] = \perp$, set $\mathsf{actual}[ssid_3] = \mathsf{actual}[ssid_1] + \mathsf{actual}[ssid_2]$ and send the message $(\mathsf{add} - \mathsf{recorded}, sid, ssid_1, ssid_2, ssid_3)$ to all P and $\mathcal{S}$.
- **Open:** Upon receiving $(\mathsf{open}, sid, ssid)$ from all parties P, if $\mathsf{actual}[ssid] = \boldsymbol{x}_{ssid} \neq \perp$, send $(\mathsf{open}, sid, ssid, \boldsymbol{x}_{ssid})$ to $\mathcal{S}$. If $\mathcal{S}$ answers with $(\mathsf{open}, sid, ssid, \boldsymbol{x}'_{ssid})$, send $(\mathsf{open}, sid, ssid, \boldsymbol{x}'_{ssid})$ to all parties in P.
- **Verify:** Upon receiving a message (verify, sid) from all parties in P, let $ssid_1, \ldots, ssid_o$ be the $ssid$s of opened commitments (*i.e.* for which $(\mathsf{open}, sid, ssid, \boldsymbol{x}'_{ssid})$ messages were sent). For $ssid \in \{ssid_1, \ldots, ssid_o\}$, set $b = 1$ if $\mathsf{actual}[ssid] = \boldsymbol{x}'_{ssid}$ or $b = 0$ if not, and send $(\mathsf{verify}, sid, ssid, b)$ to every party in P.

Fig. 9. Functionality for additively homomorphic commitments with multiple senders.

underlying multi-receiver commitment scheme) compute the same linear combination on its own "shares" of the joint commitment. Opening a joint commitment works by having each party open their individual commitments, allowing everybody to compute the joint commitment as the sum of the opened messages. Using standard tricks, these joint random commitments can be easily turned into commitments to arbitrary messages.

Security Analysis. To verify correctness, notice that Π_{MSAHCOM} computes a random commitment identified by $ssid$ as a commitment to $\sum_{i \in [t]} \mathsf{raw}^i[ssid]$, where $\mathsf{raw}^i[ssid]$ is supposed to be the value obtained by P_i from $\mathcal{F}^i_{\text{AHCOM}}$. In the verification procedure, all parties obtain $\boldsymbol{x}_j$ for $j \in [t]$ directly from $\mathcal{F}^j_{\text{AHCOM}}$, being able to verify that the previously opened commitments are indeed valid. If a commitment identified by $ssid$ is set to an arbitrary message $\boldsymbol{y}$, the sender P_j holding $\boldsymbol{y}$ broadcasts $\boldsymbol{w} = \boldsymbol{y} - \sum_{i \in [t]} \mathsf{raw}^i[ssid]$, which also allows all parties to retrieve $\boldsymbol{y}$ when values $\mathsf{raw}^i[ssid]$ are released and to verify the correctness of this opening when $\boldsymbol{x}_j$ (corresponding to $\mathsf{raw}^j[ssid]$) are revealed. Notice that

Protocol Π_{MSAHCOM}

Given a set of parties $P = \{P_1, \ldots, P_t\}$, for each party $P_i \in P$, Π_{MSAHCOM} uses an instance of $\mathcal{F}_{\text{AHCOM}}$ denoted as $\mathcal{F}^i_{\text{AHCOM}}$ where P_i is the sender with a set of receivers $V_i = P \setminus P_i$. Parties in $P = \{P_1, \ldots, P_t\}$ interact with each other and with $\mathcal{F}^1_{\text{AHCOM}}, \ldots, \mathcal{F}^t_{\text{AHCOM}}$, proceeding as follows:

1. **Commit** On input $(\mathsf{commit}, sid, ssid, \mathcal{I})$ where $\mathcal{I} = \{ssid_1, \ldots, ssid_\gamma\}$ each party $P_i \in P$, for $ssid \in \mathcal{I}$, sends $(\mathsf{commit}, sid, ssid, P_i, V_i)$ to $\mathcal{F}^i_{\text{AHCOM}}$, receiving as answer $(\mathsf{receipt}, sid, ssid, P_i, V_i, \boldsymbol{x}_{ssid})$ and setting $\mathsf{raw}^i[ssid] = \boldsymbol{x}_{ssid}$ and $\mathsf{actual}^i[ssid] = \perp$.
2. **Input** On input $(\mathsf{input}, sid, ssid, \boldsymbol{y})$ for P_i and input $(\mathsf{input}, sid, ssid, P_j)$ for every P_j for $j \neq i$, parties P proceed as follows:
   (a) For every $j \in [t], j \neq i$, P_j aborts if $\mathsf{actual}^j[ssid] \neq \perp$. Otherwise, P_j sends $(sid, ssid, \mathsf{raw}^j[ssid])$ to P_i.
   (b) Upon receiving $(sid, ssid, \mathsf{raw}^j[ssid])$ from P_j for every $j \in [t], j \neq i$, P_i sets $\boldsymbol{x} = \sum_{j \in [t]} \mathsf{raw}^j[ssid]$, $\boldsymbol{w} = \boldsymbol{y} - \boldsymbol{x}$, $\mathsf{actual}^i[ssid] = \boldsymbol{w}$ and broadcasts $(sid, ssid, P_i, \boldsymbol{w})$.
   (c) Upon receiving $(sid, ssid, P_i, \boldsymbol{w})$, every party $P_j \in P$ sets $\mathsf{actual}^j[ssid] = \boldsymbol{w}$.
3. **Random:** On input $(\mathsf{random}, sid, ssid)$, if $\mathsf{actual}^i[ssid] = \perp$, each party $P_i \in P$ sets $\mathsf{actual}^i[ssid] = \mathbf{0}^k$.
4. **Addition**: On input $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3)$, if $\mathsf{actual}^i[ssid_1] \neq \perp$, $\mathsf{actual}^i[ssid_2] \neq \perp$ and $\mathsf{actual}^i[ssid_3] = \perp$, every party $P_i \in P$ sets $\mathsf{actual}^i[ssid_3] = \mathsf{actual}^i[ssid_1] + \mathsf{actual}^i[ssid_2]$ and sends $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_i, V_i)$ to $\mathcal{F}^i_{\text{AHCOM}}$. All parties proceed after receiving $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_i, V_i, \mathsf{success})$ from $\mathcal{F}^i_{\text{AHCOM}}$.
5. **Open:** On input $(\mathsf{open}, sid, ssid)$, each $P_i \in P$ broadcasts $(sid, ssid, \mathsf{raw}^i[ssid])$. Upon receiving $(sid, ssid, \mathsf{raw}^j[ssid])$ for $j \in [t], j \neq i$, each party $P_i \in P$ computes $\boldsymbol{x}' = \mathsf{actual}^i[ssid] + \sum_{j \in [t]} \mathsf{raw}^j[ssid]$ and outputs $(sid, ssid, \boldsymbol{x}')$.
6. **Verify:** On input (verify, sid), let $ssid_1, \ldots, ssid_o$ be the $ssid$s of opened commitments (*i.e.* for which $(\mathsf{open}, sid, ssid)$ inputs were received), every $P_i \in P$ sends $(\mathsf{reveal}, sid, ssid_1, \ldots, ssid_o)$ to $\mathcal{F}^i_{\text{AHCOM}}$. For every $ssid \in \{ssid_1, \ldots, ssid_o\}$, upon receiving $(\mathsf{reveal}, sid, ssid, P_j, V_j, \boldsymbol{x_j})$ for $j \in [t], j \neq i$, each party $P_i \in P$ sets $\boldsymbol{x_i} = \mathsf{raw}^i[ssid]$, computes $\boldsymbol{x} = \mathsf{actual}^i[ssid] + \sum_{j \in [t]} \boldsymbol{x_j}$, sets $b = 1$ if $\boldsymbol{x}' = \boldsymbol{x}$ (where $\boldsymbol{x}'$ is the value previously opened) or $b = 0$ if not, and outputs $(\mathsf{verify}, sid, ssid, b)$.

Fig. 10. Protocol Π_{MSAHCOM}

addition are simply computed by adding the $\mathsf{actual}^i[ssid]$ vectors and, since all of these vectors are linear combinations of themselves, opening and verification of a result addition works the same way as for the other commitments.

Theorem 5. *Protocol Π_{MSAHCOM} UC realizes $\mathcal{F}_{\text{MSAHCOM}}$ in the $\mathcal{F}_{\text{AHCOM}}$-hybrid model with statistical security against a static adversary. Formally, there exists a simulator $\mathcal{S}$ such that for every static adversary $\mathcal{A}$, and any environment $\mathcal{Z}$ the following holds:* $\mathsf{IDEAL}_{\mathcal{F}_{\text{MSAHCOM}}, \mathcal{S}, \mathcal{Z}} \approx_s \mathsf{HYBRID}^{\mathcal{F}_{\text{AHCOM}}}_{\Pi_{\text{MSAHCOM}}, \mathcal{A}, \mathcal{Z}}$.

Proof (Sketch). Notice that Π_{MSAHCOM} only performs operations with random values obtained from $\mathcal{F}^i_{\text{AHCOM}}$. Hence, upon learning the opening of any com-

mitment from $\mathcal{F}_{\text{MSAHCOM}}$, the simulator can simply cheat in the openings of random values from the emulated $\mathcal{F}^i_{\text{AHCOM}}$ in order to equivocate a commitment. Similarly, if it needs to extract any commitment done in Π_{MSAHCOM}, the simulator can compute it from the messages sent by the adversary in the protocol and the messages the adversary obtains from the emulated $\mathcal{F}^i_{\text{AHCOM}}$.

Efficiency. Notice that our construction of Π_{MSAHCOM} using $\mathcal{F}_{\text{AHCOM}}$ as a black box actually communicates more bits than necessary. In Π_{MSAHCOM}'s opening phase, all parties broadcast the messages in commitments generated by $\mathcal{F}_{\text{AHCOM}}$ and, later on, verify these openings by opening the commitments through $\mathcal{F}_{\text{AHCOM}}$, sending the same messages again. If instantiated with Π_{AHCOM}, our construction can be made more efficient by having the parties broadcast columns $\mathbf{A}_0[\cdot, j], \mathbf{A}_1[\cdot, j]$ (Step 1 of Π_{AHCOM}'s opening phase) during the opening phase of Π_{MSAHCOM}. Later on, for verification, the parties only need to execute the remaining steps of the opening phase of Π_{AHCOM} in order to verify that the columns they have previously obtained are actually valid. In a setting with t parties, our protocol only requires t individual multi-receiver commitments, where the construction of [21] requires t^2 two-party commitments. Their constructions also require extra communication in the order of $O(skt^2)$ for generating a batch of m commitments, where s is the security parameter and k is the message length. Moreover, instantiating the construction of [21] with the previously best two-party additively homomorphic commitments [16] implies a high cost of nt^2 OTs for the setup phase (with an underlying $[n, k, s]$ code) and extra communication in the order of $O(nmt^2)$ bits for generating a batch of m commitments to random messages. On the other hand, our construction instantiated with protocol Π_{AHCOM} can do the same with nt calls to $\mathcal{F}_{\text{COM}}$ (which can be instantiated much cheaper than an OT by calling a random oracle and sending its output) and extra communication in the order of $O(smt)$ bits. In the opening phase, the construction of [21] requires communication in the order of $O(nt^2)$ bits, while our construction only requires communication in the order of $O(nt)$ bits, assuming broadcast channels.

6.2 Insured MPC

Recently, Andrychowicz et al. [2] started a line of work [6,26,7,4] that deals with the problem of fairness in multiparty computation by combining MPC protocols with cryptocurrencies. The main idea is to provide financial incentives for the parties to act honestly. In a nutshell, each party provides a security deposit before the protocol execution or right before the outputs are revealed. After that, the protocol is executed and if no problem happens, then the security deposits are reimbursed. On the other hand, if some problem happens, the security deposit of the parties who misbehaved/aborted is used to compensate the remaining parties. This combination of MPC and cryptocurrency techniques also allows to have both inputs and outputs consisting of both data and monetary assets and distribute the funds according to the output of the computation.

The most efficient solution to date, due to Baum et al. [4], uses a publicly verifiable additively homomorphic multi-receiver commitment scheme as a central building block. By combining such commitment scheme with a smart contract, an authenticated bulletin board, and a MPC scheme that output verifiably secret shared outputs, they obtained an efficient MPC protocol with public detection of cheating behavior that financially punishes misbehaving parties. Nevertheless, the main bottleneck of their protocol is the multi-party commitment scheme, as its complexity grows quadratically in the number of parties. With our techniques it is possible to greatly improve the performance of publicly verifiable additively homomorphic multi-receiver commitments.

The functionality for publicly verifiable additively homomorphic commitment $\mathcal{F}_{\text{PVHCOM}}$ is described in the full version [18] and the set of external verifiers U is allowed to be dynamic by adding procedures for registering and deregistering parties following the approach of Badertscher et al. [3]. Assuming that the underlying commitment protocol Π_{COM} used as a building block is publicly verifiable, Protocol Π_{AHCOM} is trivially publicly verifiable when all the messages are posted to an authenticated bulletin board, straightforwardly realizing functionality $\mathcal{F}_{\text{PVHCOM}}$. The "canonical" random oracle commitment scheme (that realizes $\mathcal{F}_{\text{COM}}$ in the programmable Global Random Oracle model without extra computational assumptions according to a recent result by Camenisch *et al.* [11]) is a clear example of a scheme that is publicly verifiable when the messages are posted to an authenticated bulletin board, and Π_{AHCOM} instantiated using that commitment scheme can be used to remarkably improve the performance of publicly verifiable additively homomorphic commitments and consequently of the Insured MPC protocol of Baum et al. [4]. The efficiency improvements achieved in this application are similar to those of the Committed MPC case, since the previously best publicly verifiable multi-receiver additively homomorphic commitment protocol of [4] has a very similar structure to the commitment protocol of [21].

References

1. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 2087–2104. ACM Press, October / November 2017.
2. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458. IEEE Computer Society Press, May 2014.
3. Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 324–356. Springer, Heidelberg, August 2017.
4. Carsten Baum, Bernardo David, and Rafael Dowsley. Insured mpc: Efficient secure multiparty computation with punishable abort. Cryptology ePrint Archive, Report 2018/942, 2018. `https://eprint.iacr.org/2018/942`.

5. Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 37–56. Springer, Heidelberg, August 1990.
6. Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439. Springer, Heidelberg, August 2014.
7. Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 410–440. Springer, Heidelberg, December 2017.
8. Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Analysis and improvement of Lindell's UC-secure commitment schemes. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 534–551. Springer, Heidelberg, June 2013.
9. Luís T. A. N. Brandão. Very-efficient simulatable flipping of many coins into a well - (and a new universally-composable commitment scheme). In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 297–326. Springer, Heidelberg, March 2016.
10. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg 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.
11. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, April / May 2018.
12. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
13. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.
14. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.
15. Ignacio Cascudo. On squares of cyclic codes. *IEEE Transactions on Information Theory*, 65(2):1034–1047, 2019.
16. Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, and Jesper Buus Nielsen. Rate-1, linear time and additively homomorphic UC commitments. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 179–207. Springer, Heidelberg, August 2016.
17. Ignacio Cascudo, Ivan Damgård, Bernardo Machado David, Irene Giacomelli, Jesper Buus Nielsen, and Roberto Trifiletti. Additively homomorphic UC commitments with optimal amortized overhead. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 495–515. Springer, Heidelberg, March / April 2015.
18. Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, Rafael Dowsley, and Irene Giacomelli. Efficient UC commitment extension with homomorphism for free (and applications) [full version]. Cryptology ePrint Archive, Report 2018/983, 2018. `https://eprint.iacr.org/2018/983`.

19. Ivan Damgård, Bernardo Machado David, Irene Giacomelli, and Jesper Buus Nielsen. Compact VSS and efficient homomorphic UC commitments. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 213–232. Springer, Heidelberg, December 2014.
20. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.
21. Tore K. Frederiksen, Benny Pinkas, and Avishay Yanai. Committed MPC - maliciously secure multiparty computation from homomorphic commitments. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 587–619. Springer, Heidelberg, March 2018.
22. Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. On the complexity of additively homomorphic UC commitments. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 542–565. Springer, Heidelberg, January 2016.
23. Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 537–556. Springer, Heidelberg, May 2013.
24. Juan A. Garay, Yuval Ishai, Ranjit Kumaresan, and Hoeteck Wee. On the complexity of UC commitments. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 677–694. Springer, Heidelberg, May 2014.
25. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008.
26. Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.
27. Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 446–466. Springer, Heidelberg, May 2011.
28. Hugues Randriambololona. Asymptotically good binary linear codes with asymptotically good self-intersection spans. *IEEE Trans. Information Theory*, 59(5):3038–3045, 2013.
29. Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016.
30. Salil P. Vadhan and Colin Jia Zheng. Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 817–836. ACM Press, May 2012.
31. Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.