Additively Homomorphic UC Commitments with Optimal Amortized Overhead

Ignacio Cascudo, Ivan Damgård, Bernardo David, Irene Giacomelli, Jesper Buus Nielsen, and Roberto Trifiletti *

Dept. of Computer Science, Aarhus University
{ignacio,ivan,bernardo,giacomelli,jbn,roberto}@cs.au.dk

Abstract. We propose the first UC secure commitment scheme with (amortized) computational complexity linear in the size of the string committed to. After a preprocessing phase based on oblivious transfer, that only needs to be done once and for all, our scheme only requires a pseudorandom generator and a linear code with efficient encoding. We also construct an additively homomorphic version of our basic scheme using VSS. Furthermore we evaluate the concrete efficiency of our schemes and show that the amortized computational overhead is significantly lower than in the previous best constructions. In fact, our basic scheme has amortised concrete efficiency comparable with previous protocols in the Random Oracle Model even though it is constructed in the plain model.

1 Introduction

A commitment scheme is a very basic but nevertheless extremely powerful cryptographic primitive. Intuitively, a commitment scheme is a digital equivalent of a secure box: it allows a prover P to commit to a secret s by putting it into a locked box and giving it to a verifier V. Since the box is locked, V does not learn s at commitment time and we say the commitment is *hiding*. Nevertheless, P can later choose to give V the key to the box to let V learn s. Since P gave away the box, he cannot change his mind about s after commitment time and we say the commitment is *binding*.

Commitment schemes with stand-alone security (i.e., they only have the binding and hiding properties) can be constructed from any one-way

* The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation and from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council within which part of this work was performed. Partially supported by Danish Council for Independent Research via DFF Starting Grant 10-081612. Partially supported by the European Research Commission Starting Grant 279447.

function and already this most basic form of commitments implies zero-knowledge proofs for all NP languages. Commitments with stand-alone security can be very efficient as they can be constructed from cheap symmetric cryptography such as pseudorandom generators [Nao91].

However, in many cases one would like a commitment scheme that composes well with other primitives, so that it can be used as a secure module that will work no matter which context it is used in. The strongest form of security we can ask for here is UC security [Can01]. UC commitments cannot be constructed without set-up assumptions such as a common reference string [CF01]. On the other hand, a construction of UC commitment in such models implies public-key cryptography [DG03] and even multiparty computation [CLOS02] (but see [DNO10] for a construction based only on one-way functions, under a stronger set-up assumption).

With this in mind, it is not surprising that constructions of UC commitments are significantly less efficient than those of stand-alone secure commitments. Until recently, the most efficient UC commitment schemes were based on the DDH assumption and required several exponentiations in a large group [Lin11,BCPV13]. Therefore, even though the communication complexity for committing to k strings was $O(k)$, the computational complexity was typically $\Omega(k^3)$.

However, in [DDGN14] and independently in [GIKW14], it was observed that even though we cannot build UC commitments without using public-key technology, we can still hope to confine the use of it to a once-and-for-all set-up phase, the cost of which can then be amortized over many uses of the commitment scheme.

While [GIKW14] focused on the rate of the commitment scheme, [DDGN14] concentrated on the computational complexity. More specifically, a UC commitment scheme was proposed based on the following idea: the committer will secret-share the string s to commit to using a linear secret sharing scheme (LSSS), encrypt the shares and send them to the receiver. The encryption is done in such a way that the receiver will be able to decrypt an unqualified subset (and hence will not learn s). However, the committer will not know which subset the receiver has seen. We can achieve this efficiently using a combination of oblivious transfers (done only in a set-up phase) and a pseudorandom generator. To open s, the committer will send s and the randomness used for the sharing and the receiver can then check if the resulting shares match those he already knows. Intuitively, we can hope this will be binding because any two sets of shares for different secrets must be different in many positions (they cannot agree on any qualified subset). Furthermore since the committer does not know which subset the receiver checks, it is likely that the receiver will see a mismatch for at least one of the sets of shares.

The most natural way to construct a suitable LSSS is to use the standard construction from a linear code $\mathcal{C}$, where we choose a random codeword subject to the condition that the secret s appears in the first k coordinates and the shares are then the values appearing in the rest of the codeword. This approach requires that both C and its dual have large minimum distance. But unfortunately, all known codes with linear time encoding have very bad dual codes. Therefore, [DDGN14] resorted to using Reed-Solomon codes which gives a complexity of $O(k \log k)$ for both parties.

Our contribution. In this paper, we propose a different way to construct an LSSS from a linear code $\mathcal{C}$: we encode the secret s in $\mathcal{C}$, and then additively share each entry in the codeword to form t shares, thus we get nt shares for a code of length n. We show that already for $t \geq 2$, using this LSSS in the above template construction results in a secure UC commitment scheme. Note that the LSSS we construct is not of the usual threshold type where any sufficiently large set can reconstruct, but instead we have a more general access structure where the qualified sets are those that can get enough entries in the underlying codeword to be able to decode.

Since we can now choose $\mathcal{C}$ without any conditions on the dual code, we can plug in known constructions of codes with linear time encoding and get complexity $O(k)$ for both parties. Furthermore we show a particular instantiation of the building blocks of our basic protocol for security parameter $\tau = 60$ and message length $k = 256$ that achieves an amortized computational complexity which is 5500 times lower than in the most efficient previous constructions [BCPV13,Lin11] (see Section 6 for details on the implementation). In fact, our basic scheme achieves amortised concrete effieciency comparable to previous schemes [HM04,DSW08] in the Random Oracle Model [BR93] even though it is constructed in the plain model. Concretely, it has an amortized computational cost 41% lower than the one of [HM04].

Commitment schemes can be even more useful if they are homomorphic. An additively homomorphic commitment scheme, for instance, has the following property: from commitments to s and s', the receiver can on his own compute a commitment to $s + s'$, such that if the committer opens this new commitment, $s + s'$ (and no other information on s, s') will be revealed. Our basic construction above is not additively homomorphic. The reason is that a corrupt committer may submit sets of values in the commit phase that are not consistent sharings of any value. Nevertheless, when some of these shares are added, we may get values that do in fact form valid commitments, and this may allow the committer to cheat. To solve this problem, we start from an idea that was introduced in [GIKW14]: they construct a very compact linear verifiable secret sharing scheme (VSS) from any LSSS. The idea is now that the committer will execute the VSS "in his head" and send to the receiver the resulting views of each VSS-player, encrypted in the same way as we encrypted shares before: the receiver can decrypt some subset of the views. The receiver will now be able to execute some of those consistency checks that honest players would normally do in the VSS, and will reject if anything is wrong. The hope is that this will force the committer to submit views from a correctly executed instance of the VSS, which in particular means that the sets of shares he submits will be consistent, thus implying the additive homomorphic property.

This idea was shown to work in [DDGN14], but unfortunately the proof works only if the underlying LSSS is a threshold scheme, and our LSSS is not threshold. However, in this paper, we give a different proof showing that we do in fact get a secure commitment scheme if we choose the parameter t from our LSSS to be at least 3. This yields a UC secure and additively homomorphic commitment scheme with linear complexity, albeit with larger hidden constants than our first scheme. We also instantiate this scheme for concrete parameters, see Section 6.

It is interesting to note that there is strong relation between the way the VSS is used here and the "MPC-in-the-head" line of work [IPS09]. Roughly speaking, MPC-in-the-head is a general technique for turning a multiparty protocol into a 2-party protocol for the same purpose. A VSS is essentially a multiparty commitment scheme, so one can use the so-called IPS compiler on the VSS from [DDGN14] to get a UC secure commitment scheme. This commitment scheme is quite similar to (but not the same as) the one from [DDGN14]. Previously, the IPS compiler was only known to work for protocols with threshold security. However, our proof technique also applies to IPS, so from this point of view, our result is the first to show that the IPS compiler can also be used to transform a non-threshold multiparty protocol into a 2-party protocol. It is an interesting open problem to characterise the adversary structures for which it will work.

2 Preliminaries

2.1 Notation

We denote uniformly sampling a value r from a set D as $r \leftarrow D$ and $\{r_1, \ldots, r_n\} \leftarrow D$ indicates that we sample from D a uniformly random subset of n elements. We denote concatenation by $\|$ and vectors of elements of some field by bold symbols. For $\boldsymbol{z} \in \mathbb{F}^k$, $\boldsymbol{z}[i]$ denotes the i'th entry of the vector. We use 1-indexing, meaning that $\boldsymbol{z}[1]$ is the first element of $\boldsymbol{z}$ and we write $[n] = \{1, 2, \ldots, n\}$. We will use π_k to denote the projection that outputs the first k coordinates of a vector, *i.e.* $\pi_k(\boldsymbol{z}) = (\boldsymbol{z}[1], \ldots, \boldsymbol{z}[k])$. Finally we will denote by $\boldsymbol{e}_{k,i}$ the row vector of k components whose i-th entry is 1 while all other entries are 0 and with $\mathbf{0}_k$ the row vector of k components whose all entries are 0.

We say that a function ϵ is negligible in n if for every 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 *indistinguishable*, denoted by $X \approx Y$, if for all z it holds that $\mid Pr[X_{\kappa,z} = 1] - Pr[Y_{\kappa,z} = 1] \mid$ is negligible in κ.

2.2 Universal Composability

The results presented in this paper are proven secure in the Universal Composability (UC) framework introduced by Canetti in [Can01]. Definitions can be found in Appendix A.2 of this paper.

Adversarial Model: In this work we consider security against static adversaries, *i.e.* corruption may only take place *before* the protocols starts execution. We consider active adversaries who may deviate from the protocol in any arbitrary way.

Setup Assumption: It is known that UC commitment protocols (as well as most "interesting" functionalities) cannot be obtained in the plain model [CF01]. In order to overcome this impossibility, UC protocols require a setup assumption, that basically models a resource that is made available to all parties before execution starts. The security of our protocols is proved in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model [Can01,CLOS02], where all parties are assumed to have access to an ideal 1-out-of-2 OT functionality (see Figure 8).

Ideal Functionalities: In Section 4, we construct a simple string commitment protocol that UC-realizes the functionality $\mathcal{F}_{\mathrm{COM}}$ as presented in [CLOS02,CDD+14] and recalled here in Figure 6. In Section 5, we extend this simple scheme to allow homomorphic operations over commitments. The extended protocol UC-realizes the functionality $\mathcal{F}_{\mathrm{HCOM}}$ in Figure 7, that basically adds a command for adding two previously stored commitments and an abort command in the Commit Phase to $\mathcal{F}_{\mathrm{COM}}$. The abort is necessary to deal with inconsistent commitments that could be sent by a corrupted party. In fact, our additively homomorphic commitment protocol is constructed in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid model (*i.e.* assuming access to $(t-1)$-out-of-t OT where $t \geq 2$ is an integer parameter). Notice that $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$ is basically a special case of a k-out-of-n OT where $k = n - 1$, which can be subsequently reduced to the $\mathcal{F}_{\mathrm{OT}}$-hybrid model via standard techniques [Nao91,BCR86,NP99]. We define $\mathcal{F}_{\mathrm{OT}}$ in Figure 8 and $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$ in Figure 9 following the syntax of [CLOS02]. Notice that $\mathcal{F}_{\mathrm{OT}}$ can be efficiently UC-realized by the protocol in [PVW08], which can be used to instantiate the setup phase of our commitment protocols.

2.3 Linear Secret-Sharing Scheme

We briefly recall here the definition of linear secret-sharing scheme (LSSS) following the approach of [CDP12].

Definition 1. *A linear secret sharing scheme for N players $P_1, \ldots, P_N$ over the finite field $\mathbb{F}$ is defined by the pair $(k, \boldsymbol{M})$, where k is the length of a secret and $\boldsymbol{M}$ is a $N \times m$ matrix with entries in $\mathbb{F}$ (and $m > k$). If $k > 1$, then the scheme is called* packed. *The row number i of $\boldsymbol{M}$ is denoted by $\boldsymbol{m}_i$ and, if A is a subset of players, then $\boldsymbol{M}_A$ denotes the matrix consisting of rows $\boldsymbol{m}_i$ such that $P_i \in A$.*

In order to share a secret $\boldsymbol{s} \in \mathbb{F}^k$, the dealer of the LSSS given by $(k, \boldsymbol{M})$ takes a random column vector $\boldsymbol{f} \in \mathbb{F}^m$ such that $\pi_k(\boldsymbol{f}) = \boldsymbol{s}^\top$ and computes $\boldsymbol{c} = \boldsymbol{M} \cdot \boldsymbol{f}$. The column vector $\boldsymbol{c}$ is called the *share vector* of $\boldsymbol{s}$, and its i-th component $\boldsymbol{c}[i]$ is the share sent by the dealer to the player P_i.

Definition 2. *A subset of players A is called* unqualified *if the distribution of $\boldsymbol{M}_A \cdot \boldsymbol{f}$ is independent of $\boldsymbol{s}$, while a subset of players B is called* qualified *if $\boldsymbol{s}$ is uniquely determined from $\boldsymbol{M}_B \cdot \boldsymbol{f}$.*

It is the case that A is unqualified if and only if there exists, for each position j in $\boldsymbol{s}$, a column vector of m components $\boldsymbol{w}^{A,j}$ (called *sweeping vector*) such that $\boldsymbol{w}^{A,j} \in \ker(\boldsymbol{M}_A)$ and $\pi_k(\boldsymbol{w}^{A,j}) = \boldsymbol{e}_{k,j}^\top$. Similarly, B is qualified if and only if there exists, for each position j in $\boldsymbol{s}$, a row vector of $|B|$ components $\boldsymbol{r}_{B,j}$ (called *reconstruction vector*) such that $\boldsymbol{r}_{B,j} \cdot \boldsymbol{M}_B = \boldsymbol{e}_{m,j}$.

Given two positive integers a and b, if any subset of players A with $|A| = a$ is unqualified, then we say that the LSSS has a-privacy. If any subset of players B with $|B| = b$ is qualified, then we say that the LSSS has b-reconstruction.

Example 1. The *additive secret-sharing scheme* for N players over $\mathbb{F}$ is the linear secret-sharing scheme where in order to share a secret $s \in \mathbb{F}$ among N players, the dealer chooses random values $s_1, \ldots, s_N$ in $\mathbb{F}$ such that $\sum_{i=1}^{N} s_i = s$ and sends the value s_i to player i. It is clear that the set of all the players can reconstruct the secret from the received values, while any set of at most $N-1$ players has no information on the value s held by the dealer. With the previous notation, this LSSS can be defined by the pair $(1, \boldsymbol{M})$, where $\boldsymbol{M}$ has the following N rows: $\boldsymbol{m}_i = \boldsymbol{e}_{N,i+1}$ for $i = 1, \ldots, N-1$ and $\boldsymbol{m}_N = \boldsymbol{e}_{N,1} - \sum_{i=1}^{N-1} \boldsymbol{m}_i$.

3 Linear-Time Secret Sharing and Coding Scheme

In this section we describe the coding scheme $\mathcal{C}_t$ that stands in the core of our commitment protocols. We depart from *any* error correcting code and apply a simple transformation that yields a code that can also be seen as a linear secret sharing scheme for an specific access structure. Intuitively, this makes it possible to reveal a large fraction of a codeword generated by $\mathcal{C}_t$ without revealing any information on the encoded message.

Standard generic constructions of general linear secret sharing schemes from error correcting codes, require a code whose dual code has high minimum distance. On the other hand, our construction does not require any specific property from the underlying error correcting code. This conceptual difference is of fundamental importance for the asymptotic and concrete efficiencies of our constructions, since it allows a secret sharing scheme to be constructed from very efficient linear error correcting codes whose dual codes' minimum distance are mostly unfit for the standard generic constructions. In particular, our coding scheme $\mathcal{C}_t$ inherits the underlying code $\mathcal{C}$'s complexity. achieving linear-time encoding and/or decoding when constructed from appropriate codes [GI01,GI02,GI03,GI05,Spi96,DI14].

Intuitively, the encoding procedure $\mathsf{Enc}_t^{\mathcal{C}}$ of $\mathcal{C}_t$ first encodes a message $\boldsymbol{m}$ under the underlying code $\mathcal{C}$ obtaining a codeword $\boldsymbol{v}$. In the next step, each element of $\boldsymbol{v}$ is secret shared into t shares under a simple additive secret sharing scheme (*i.e.* taking random vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t$ such that $\sum_{i=1}^{t} \boldsymbol{v}_i = \boldsymbol{v}$). The final codeword $\boldsymbol{c}$ is defined as $\boldsymbol{c} = (\boldsymbol{v}_1[1], \ldots, \boldsymbol{v}_t[1], \ldots, \boldsymbol{v}_1[n], \ldots,$

$\boldsymbol{v}_t[n])^\top$, *i.e.*, each t successive elements of $\boldsymbol{c}$ sum up to the corresponding element of $\boldsymbol{v}$. The decoding procedure $\mathsf{Dec}_t^{\mathcal{C}}$ basically reconstructs each element of $\boldsymbol{v}$ from $\boldsymbol{c}$ and then uses the decoding algorithm of $\mathcal{C}$ to decode $\boldsymbol{v}$ into the original message $\boldsymbol{m}$. Figure 5 illustrates the inner workings of our coding scheme.

Notice that only the encoding procedure $\mathsf{Enc}_t^{\mathcal{C}}$ is used in the actual commitment schemes, while the decoding procedure $\mathsf{Dec}_t^{\mathcal{C}}$ is used in the simulators. Moreover, $\mathcal{C}_t$ basically applies a linear transformation on codewords generated by the underlying code $\mathcal{C}$, since $\mathsf{Enc}_t^{\mathcal{C}}$ uses a LSSS to divide each component of the codeword into t shares. Hence, if $\mathcal{C}$ is linear, so is $\mathcal{C}_t$. Finally, we show in Remark 1 that $\mathsf{Enc}_t^{\mathcal{C}}$ itself can be seen as a LSSS. Intuitively, after a message $\boldsymbol{m}$ is encoded through $\mathsf{Enc}_t^{\mathcal{C}}$, an element $\boldsymbol{v}[i]$ of the underlying codeword can only be recovered if all shares $\boldsymbol{v}_1[i], \dots, \boldsymbol{v}_t[i]$ are known. Hence, no information on $\boldsymbol{m}$ is revealed as at least one share is missing for every underlying codeword element.

Before we formally outline the coding scheme $\mathcal{C}_t$ Figure 1, we need to define the auxiliary functions Σ_t and Λ_t:

- $\Sigma_t : \mathbb{F}^n \longrightarrow \mathbb{F}^{tn}$ is a randomized function that takes as input a row vector $\boldsymbol{v}$ in $\mathbb{F}^n$ and does the following: sample $\boldsymbol{v}_1, \dots, \boldsymbol{v}_{t-1} \leftarrow \mathbb{F}^n$ and compute $\boldsymbol{v}_t = \boldsymbol{v} - (\boldsymbol{v}_1 + \cdots + \boldsymbol{v}_{t-1})$. For $j = 1, \dots, n$, define $\boldsymbol{w}_j = \|_{i=1}^{t} \boldsymbol{v}_i[j] = (\boldsymbol{v}_1[j], \dots, \boldsymbol{v}_t[j])$ and set $\Sigma_t(\boldsymbol{v}) = \|_{j=1}^{n} \boldsymbol{w}_j = (\boldsymbol{w}_1, \dots, \boldsymbol{w}_n)$. Note that this means each consecutive t-tuple of $\Sigma_t(\boldsymbol{v})$ sums to the corresponding element in the vector $\boldsymbol{v}$.
- $\Lambda_t : \mathbb{F}^{tn} \to \mathbb{F}^n$ takes as input a vector $\boldsymbol{h}$ and adds each consecutive t components of $\boldsymbol{h}$. That is, $\Lambda_t(\boldsymbol{h})$ gives as output the row vector in $\mathbb{F}^n$ whose i'th component is $\sum_{j=1}^{t} \boldsymbol{h}[(i-1)t + j]$. Note that $\Lambda_t\left(\Sigma_t(\boldsymbol{m})\right) = \boldsymbol{m}$.

Remark 1. It is possible to see the entire encoding procedure $\mathsf{Enc}_t^{\mathcal{C}}$ as a LSSS for $N = tn$ players: let $\boldsymbol{C} \in \mathbf{Mat}_{n \times k}$ be the transpose of a generator matrix for the code $\mathcal{C}$ and let $\boldsymbol{c}_j$ be its jth row, then the vector $\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m})$ can be seen as a share vector of $\boldsymbol{m} \in \mathbb{F}^k$ in the LSSS defined by the pair $(k, \boldsymbol{M}_t^{\mathcal{C}})$, where $m = k + (t-1)n$ and $\boldsymbol{M}_t^{\mathcal{C}}$ is a $N \times m$ matrix with rows given by $\boldsymbol{m}_i = \boldsymbol{e}_{m, i+k-\lfloor i/t \rfloor}$ for $i \in [nt] \setminus \{t, 2t, \dots, nt\}$ and $\boldsymbol{m}_{jt} = \left(\boldsymbol{c}_j, \mathbf{0}_{(t-1)n}\right) - \sum_{i=1}^{t-1} \boldsymbol{m}_{(j-1)t+i}$ for $j \in [n]$.

The set of tn players can be divided in n groups of t players each: define $T_j = \left\{P_{(j-1)t+1}, \dots, P_{jt}\right\}$ for all $j \in [n]$. Thus we can rephrase the encoding procedure $\mathsf{Enc}_t^{\mathcal{C}}$ for a vector $\boldsymbol{m} \in \mathbb{F}^k$ as: first compute the codeword $\boldsymbol{v} = \mathcal{C}(\boldsymbol{m})$ and then, for all $j \in [n]$, share the component $\boldsymbol{v}[j]$ be-

Coding Scheme $\mathcal{C}_{\mathbf{t}}$

Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear error correcting code over a field $\mathbb{F}$ of dimension k, length n and minimum distance d, and let $t \geq 2$ be a fixed integer. Let $\boldsymbol{m}$ be a row vector in $\mathbb{F}^k$and $\boldsymbol{c}$ be a column vector in $\mathbb{F}^{tn}$ The coding scheme is composed by the pair of algorithms $\left(\mathsf{Enc}_t^{\mathcal{C}}, \mathsf{Dec}_t^{\mathcal{C}}\right)$ described as follows:

– $\mathsf{Enc}_{\mathbf{t}}^{\mathcal{C}}(\boldsymbol{m})$: the encoding procedure $\mathsf{Enc}_t^{\mathcal{C}} : \mathbb{F}^k \to \mathbb{F}^{tn}$ takes as input a message $\boldsymbol{m}$ and proceeds as follows:
  1. Encode $\boldsymbol{m}$ using $\mathcal{C}$, thus obtaining $\boldsymbol{v} = \mathcal{C}(\boldsymbol{m}) \in \mathbb{F}^n$.
  2. Use the randomized function $\Sigma_t(\boldsymbol{v})$ to additively secret share each component of the codeword $\boldsymbol{v}$ into t shares. Output the column vector $\boldsymbol{c} = \Sigma_t(\boldsymbol{v})^\top$. When we need to remember the randomness used in Σ_t, we will write $\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m}; \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$.

Let $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$ and let $\mathcal{D} : \mathbb{F}^n \to \mathbb{F}^n \cup \{\perp\}$ be a τ-bounded decoding algorithm for the underlying code $\mathcal{C}$. That is, $\mathcal{D}$ either decodes a received word $\boldsymbol{r}$ into the unique codeword $\boldsymbol{c} \in \mathcal{C}$ at distance not more than τ from $\boldsymbol{r}$ (if such codeword exists) or indicates that no such codeword exists, declaring a decoder failure.

– $\mathsf{Dec}_{\mathbf{t}}^{\mathcal{C}}(\boldsymbol{c})$: the decoding procedure $\mathsf{Dec}_t^{\mathcal{C}} : \mathbb{F}^{tn} \to \mathbb{F}^k \cup \{\perp\}$ takes as input a codeword $\boldsymbol{c}$ and proceeds as follows:
  1. Compute $\Lambda_t(\boldsymbol{c})$ to obtain a vector $\boldsymbol{v}' \in \mathbb{F}^n$.
  2. Decode $\boldsymbol{v}'$ using the decoding algorithm $\mathcal{D}$ for the underlying code $\mathcal{C}$. If $\mathcal{D}$ fails, output $\perp$. Otherwise output $\boldsymbol{m} = \mathcal{C}^{-1}(\mathcal{D}(\boldsymbol{v}'))$.

Fig. 1. Coding Scheme $\mathcal{C}_t$

tween the players in T_j using the additive LSSS for t players (see Example 1). From the $(t-1)$-privacy property of the additive LSSS, it follows that any subset of players $A \subseteq \{P_1, \ldots, P_n\}$ such that $|A \cap T_j| \leq t-1$ for all $j \in [n]$ is unqualified for the scheme $(k, \boldsymbol{M}_t^{\mathcal{C}})$. Instead, if $B \subseteq \{P_1, \ldots, P_n\}$ satisfies $B \cap T_j = T_j$ for at least $n-(d-1)$ indices j, then it is a qualified set for $(k, \boldsymbol{M}_t^{\mathcal{C}})$. Indeed, the players in B can compute at least $n-(d-1)$ components of the codeword $\boldsymbol{v}$ and then they can apply an erasure correction algorithm for $\mathcal{C}$ and recover $\boldsymbol{m}$. In particular if $|B| \geq nt-(d-1)$, then B is qualified.

4 Basic Construction

In this section we present our basic commitment scheme. We will work in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid model ($t$ being a fixed integer greater or equal than 2) and we will phrase our protocol in terms of a Setup and an Online phase. This decoupling is motivated by the fact that the Setup phase can be run at any time and independently of the inputs of the parties. Once the Setup phase is completed, polynomially many commitments can be executed in the Online phase, when the inputs are known. Moreover, the Setup phase is also completely independent of the number of commitments executed in the Online phase. Finally our scheme is based on a $[n, k, d]$ linear error correcting code $\mathcal{C}$ over $\mathbb{F}$ used in the encoding procedure $\mathsf{Enc}_t^{\mathcal{C}}$ defined in Figure 1 (we consider $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$ the security parameter).

A commitment to a message $\boldsymbol{m} \in \mathbb{F}^k$ will be obtained by sending to the receiver P_r a subset of components (*watch-list*) of the vector $\boldsymbol{w} = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m})$ computed by the sender P_s. The watch-list has to be chosen in such a way that the components of $\boldsymbol{w}$ contained in it give no information on the message $\boldsymbol{m}$ (hiding property). To open the commitment, the sender P_s has to send to the receiver both $\boldsymbol{m}$ and the randomness used in the procedure $\mathsf{Enc}_t^{\mathcal{C}}$, so that the receiver can compute by itself $\boldsymbol{w}$ and check if it is consistent with the components it already knows from the watch-list. If we design the protocol in such a way that the sender doesn't know which components the receiver will check, then, since P_s can not change the message it committed to without changing a substantial amount of entries, P_r will see a mismatch and catch the cheating opening with high probability (binding property).

The watch-list mechanism is created in the Setup phase. The idea is that the sender and the receiver run n $(t-1)$-out of t OTs on n groups of tn seeds for a PRG, in such a way that for each group the verifier will know only $(t-1)$ of the seeds chosen by the sender. The expanded strings produced by the PRG are used to form a matrix $\boldsymbol{Y}$. After that, in the Online phase, for each new commitment, the sender choses a new column $\boldsymbol{y}^{\eta}$ in $\boldsymbol{Y}$ and use it as one-time pad for sending to P_r the encoding $\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m})$. This will allow the receiver P_s to view $(t-1)n$ entries of the encodings without the sender knowing which these entries are. Furthermore, in this way we can allow many commitments while using the OT-functionality only once. For every new commitment, the sender and receiver can obtain new one-time pads for the watch-list by simply expanding the PRG seeds into a larger pseudorandom string up to a polynomially bounded length.

Statistical binding property: if the sender wants to open two different messages $\boldsymbol{m}$ and $\boldsymbol{m}'$ for the same commitment $(\eta, \boldsymbol{c})$, then it has to produce randomness consistent with two vectors $\boldsymbol{w}$ and $\boldsymbol{w}'$ such that $\mathcal{C}(\boldsymbol{m}) = \Lambda_t(\boldsymbol{w})$ and $\mathcal{C}(\boldsymbol{m}') = \Lambda_t(\boldsymbol{w}')$. Since the code has minimal distance d and $d \geq 2\tau + 1$, at least one of the two different codewords $\Lambda_t(\boldsymbol{w})$ and $\Lambda_t(\boldsymbol{w}')$ is at distance strictly greater than τ from $\Lambda_t(\boldsymbol{c} - \boldsymbol{y}^{\eta})$ (Hamming

Protocol $\mathbf{\Pi}_{\text{COM}}$ in the $\mathcal{F}_{\text{OT}}^{t-1,t}$-hybrid model

Let $G : \{0,1\}^{l'} \to \{0,1\}^{l}$ be a pseudorandom generator, $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear error correction code over $\mathbb{F}$ and $t \geq 2$ a fixed integer. The procedure $\mathsf{Enc}_t^{\mathcal{C}}$ is defined in Figure 1.

A sender P_s and receiver P_r interact between themselves and with $\mathcal{F}_{\text{OT}}^{t-1,t}$ as follows:

OT-Setup phase:

For $i = 1, t+1, 2t+1, \ldots, (n-1)t+1$:

1. P_s samples t strings $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_{i+t-1} \leftarrow \{0,1\}^{l'}$ and sends $(\mathsf{sender}, sid, ssid, (\boldsymbol{x}_i, \ldots, \boldsymbol{x}_{i+t-1}))$ to $\mathcal{F}_{\text{OT}}^{t-1,t}$.
2. P_r samples $\{c_1^i, \ldots, c_{t-1}^i\} \leftarrow \{0, 1, \ldots, t-1\}$ and sends $(\mathsf{receiver}, sid, ssid, c_1^i, \ldots, c_{t-1}^i)$ to $\mathcal{F}_{\text{OT}}^{t-1,t}$.
3. P_r receives $(\mathsf{received}, sid, ssid, \boldsymbol{x}_{i+c_1^i}, \ldots, \boldsymbol{x}_{i+c_{t-1}^i})$ from $\mathcal{F}_{\text{OT}}^{t-1,t}$.

Let W (*watch-list*) be the set of indices $W = \left\{i + c_1^i, \ldots, i + c_{t-1}^i \mid i = 1, t+1, 2t+1, \ldots, (n-1)t+1\right\}$ and let $\boldsymbol{Y} \in \mathbf{Mat}_{tn \times l}$ be the $tn \times l$ matrix with rows $\boldsymbol{y}_j$'s consisting of the row vectors $G(\boldsymbol{x}_j)$'s for $j = 1, \ldots, tn$. Denote by $\boldsymbol{y}^j$ the j'th column of $\boldsymbol{Y}$. P_s knows the entire matrix $\boldsymbol{Y}$, P_r knows the watch-list W and only $(t-1)n$ rows of $\boldsymbol{Y}$, but in a structured way: for each groups of t rows $\boldsymbol{y}_{jt+1}, \ldots, \boldsymbol{y}_{(j+1)t}$ it holds exactly $t-1$ of those[a].

Commit phase:

1. Upon input $(\mathsf{commit}, sid, ssid, P_s, P_r, \boldsymbol{m})$ for $\boldsymbol{m} \in \mathbb{F}^k$, P_s samples $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1} \leftarrow \mathbb{F}^n$ and computes $\boldsymbol{w} = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m}; \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$. Then P_s chooses an unused column $\boldsymbol{y}^\eta$ from the matrix $\boldsymbol{Y}$ defined in the Setup phase, computes $\boldsymbol{c} = \boldsymbol{w} + \boldsymbol{y}^\eta$ and sends $(sid, ssid, \eta, \boldsymbol{c})$ to P_r.
2. P_r stores $(sid, ssid, \eta, \boldsymbol{c})$ and outputs $(\mathsf{receipt}, sid, ssid, P_s, P_r)$.

Open phase:

1. Upon input $(\mathsf{reveal}, sid, ssid, P_s, P_r)$, P_s sends $(sid, ssid, \boldsymbol{m}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$ to P_r.
2. P_r receives $(sid, ssid, \boldsymbol{m}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$, computes $\boldsymbol{w} = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m}; \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$ and checks if $\boldsymbol{w}[i] + \boldsymbol{y}^\eta[i] = \boldsymbol{c}[i]$ for all $i \in W$. If this check fails P_r rejects the opening and halts. Otherwise P_r outputs $(\mathsf{reveal}, sid, ssid, P_s, P_r, \boldsymbol{m})$.

[a] We remark that the parties do not need to hold the entire matrices at any one point in time, but can generate it on demand using an appropriate pseudorandom generator.

Fig. 2. Protocol $\mathbf{\Pi}_{\text{COM}}$

distance). Assume w. l. o. g. that $d_{\text{Ham}}(\Lambda_t(\boldsymbol{w}), \Lambda_t(\boldsymbol{c} - \boldsymbol{y}^\eta)) \geq \tau + 1$, then in $\boldsymbol{w} - \boldsymbol{c} + \boldsymbol{y}^\eta$ there are at least $\tau + 1$ groups of consecutive entries in which at least one entry is not zero. Since the receiver checks $t - 1$ entries chosen at random in each group, the probability that he doesn't see any mismatch is at most $\left(\frac{1}{t}\right)^{\tau+1}$.

Computational hiding property: from the security of the PRG G, we can claim that the receiver knows only $t-1$ entries in each group of consecutive entries of $\boldsymbol{w}^\eta = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m})$. That is, P_r knows only $t - 1$ shares of each component of the codeword $\mathcal{C}(\boldsymbol{m})$. Thus, the hiding property follows from the $(t - 1)$-privacy property of the additive secret-sharing scheme for t players used to share each component of the codeword $\mathcal{C}(\boldsymbol{m})$.

The protocol $\mathbf{\Pi}_{\text{COM}}$ UC-realizes the ideal functionality $\mathcal{F}_{\text{COM}}$ in the $\mathcal{F}_{\text{OT}}^{t-1,t}$-hybrid model, as stated in the following two propositions. See Appendix A.3 for the proofs.

Proposition 1 (Statistical Binding Property). *Let $G : \{0,1\}^{l'} \to \{0,1\}^l$ be a pseudorandom generator and $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a $[n, k, d]$ error correction code over $\mathbb{F}$. For every static active adversary $\mathcal{A}$ corrupting only P_s in the $\mathcal{F}_{\text{OT}}^{t-1,t}$-hybrid execution of $\mathbf{\Pi}_{\text{COM}}$ and for every environment[1] $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\text{COM}},\mathcal{S},\mathcal{Z}} \approx \mathsf{HYBRID}_{\mathbf{\Pi}_{\text{COM}},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\text{OT}}^{t-1,t}}$$

where the security parameter is $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Proposition 2 (Computational Hiding Property). *Let $G : \{0,1\}^{l'} \to \{0,1\}^l$ be a pseudorandom generator and $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a $[n, k, d]$ error correction code over $\mathbb{F}$. For every static active adversary $\mathcal{A}$ corrupting only P_r in the $\mathcal{F}_{\text{OT}}^{t-1,t}$-hybrid model execution of $\mathbf{\Pi}_{\text{COM}}$ and for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\text{COM}},\mathcal{S},\mathcal{Z}} \approx \mathsf{HYBRID}_{\mathbf{\Pi}_{\text{COM}},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\text{OT}}^{t-1,t}}$$

where the security parameter is $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

[1] Note that in the proof of Proposition 1 the requirement for the environment to be polynomial-time is not necessary. Indeed the proof holds for any environment that interacts with each system only a polynomial number of times.

5 Additive Homomorphic Property

Notice that in the protocol $\mathbf{\Pi}_{\mathrm{COM}}$ a commitment $(i, \boldsymbol{c})$ may be accepted in the Open phase by an honest receiver even if $\Lambda_t(\boldsymbol{w}^i)$ is not a codeword, but it is near enough to a codeword. More precisely, if a cheating sender computes $\boldsymbol{w}^i$ in such a way that $\Lambda_t(\boldsymbol{w}^i) = \mathcal{C}(\boldsymbol{m}) + \boldsymbol{e}$ for some error vector $\boldsymbol{e}$ with Hamming weight equal to e, then an honest receiver will accept the commitment $(i, \boldsymbol{c})$ for the message $\boldsymbol{m}$ with probability equal to $\left(\frac{1}{t}\right)^e$.

Because of this, a cheating sender can setup an attack where with non negligible probability the sum of two commitments can be opened to a message that is different to the sum of the messages contained in the individual commitments. Given the vectors $\boldsymbol{m}, \boldsymbol{m}'$ and $\tilde{\boldsymbol{m}}$ where $\tilde{\boldsymbol{m}} \neq \boldsymbol{m}+\boldsymbol{m}'$, P_s can compute the vectors $\boldsymbol{e}$, $\boldsymbol{e}'$ and $\tilde{\boldsymbol{e}}$ such that $\boldsymbol{e} + \boldsymbol{e}' + \tilde{\boldsymbol{e}} = \mathcal{C}(\boldsymbol{m} + \boldsymbol{m}') - \mathcal{C}(\tilde{\boldsymbol{m}})$ and the Hamming weight of each of them is less or equal than τ (note that this is possible to achieve as long as $d \leq 3\tau$, which is not disallowed by our assumption $d \geq 2\tau + 1$). In the Commit phase the corrupted P_s defines $\boldsymbol{w} = \Sigma_t(\mathcal{C}(\boldsymbol{m}) - \boldsymbol{e})$ and $\boldsymbol{w}' = \Sigma_t(\mathcal{C}(\boldsymbol{m}') - \boldsymbol{e}')$ and sends $(\alpha, \boldsymbol{c})$ and $(\beta, \boldsymbol{c}')$, where $\boldsymbol{c} = \boldsymbol{w} + \boldsymbol{y}^\alpha$ and $\boldsymbol{c}' = \boldsymbol{w}' + \boldsymbol{y}^\beta$. Recall that Σ_t is the outer additive code in our encoding. From the above argument, in the Open phase, an honest receiver will accept $(\alpha, \boldsymbol{c})$ or $(\beta, \boldsymbol{c}')$ as commitment for $\boldsymbol{m}$ or for $\boldsymbol{m}'$ respectively, with probability strictly greater than $\left(\frac{1}{t}\right)^{\tau+1}$ in both cases. Furthermore with the same probability, P_s can also open the sum $\boldsymbol{c} + \boldsymbol{c}'$ to $\tilde{\boldsymbol{m}}$ because by construction $\boldsymbol{w} + \boldsymbol{w}' = \Sigma_t(\mathcal{C}(\tilde{\boldsymbol{m}}) + \tilde{\boldsymbol{e}})$.

While we could prevent the attack above by imposing the stronger condition $d \geq 3\tau + 1$, it is easy to see that the same problem would still apply to the additions of at least $\lceil \frac{d}{\tau} \rceil - 1$ commitments.

To deal with this problem, we need to assure that for any vector $\boldsymbol{w}$ computed by the sender in the Commit phase, it holds that $\Lambda_t(\boldsymbol{w})$ is an actual codeword. Since a correct vector $\boldsymbol{w}$ can be seen as a share-vector in the LSSS given by $(k, \boldsymbol{M}_t^{\mathcal{C}})$ (Remark 1), a standard way to achieve this guaranty is to convert $(k, \boldsymbol{M}_t^{\mathcal{C}})$ into a *verifiable secret-sharing scheme* (VSS). The latter is a secret-sharing scheme for which, together with the standard privacy property for unqualified sets of players, a stronger reconstruction property holds for the qualified sets. Indeed, in a VSS, even when the dealer is corrupted, any qualified set of honest players can determine a secret that is consistent with the share held by any honest player in the scheme. In order to obtain the additive homomorphic property for our commitment protocol, the basic idea we will use in Section 5.2 consists in forcing the sender to compute the vector $\boldsymbol{w}$ using a verifiable version of the encoding procedure $\mathsf{Enc}_t^{\mathcal{C}}$. In this way the receiver can verify that $\boldsymbol{w}$ has been properly constructed (i.e. $\Lambda_t(\boldsymbol{w})$ is a codeword) with overwhelming probability.

5.1 Packed Verifiable Secret-Sharing Scheme

In this section we recall the packed verifiable secret-sharing protocol described in [DDGN14]. We refer to the latter for the proof of the following lemmas. The protocol can be based on any linear secret-sharing scheme $(k, \boldsymbol{M})$ for N players as defined in Section 2 and it secret-shares k vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k \in \mathbb{F}^k$ in each its execution (the LSSS is over the field $\mathbb{F}$). In the following, $\boldsymbol{F}$ will be a $m \times m$ matrix with entries in $\mathbb{F}$ (m is the number of columns in $\boldsymbol{M}$) and $\boldsymbol{f}^b$ will be its the b-th column. For any index $i = 1, \ldots, N$ define the column vector $\boldsymbol{h}^i = \boldsymbol{F} \cdot \boldsymbol{m}_i^\top$ and the row vector $\boldsymbol{g}_i = \boldsymbol{m}_i \cdot \boldsymbol{F}$ (where $\boldsymbol{m}_i$ is the i-th row in $\boldsymbol{M}$). It is then clear that $\boldsymbol{m}_j \cdot \boldsymbol{h}^i = \boldsymbol{g}_j \cdot \boldsymbol{m}_i^\top$ for all $i, j \in [N]$. The VSS protocol is shown in Figure 3.

Protocol $\boldsymbol{\Pi}_{\mathrm{VSS}}$ $(\boldsymbol{M})$

1. Let $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k \in \mathbb{F}^k$ be the secrets to be shared. The dealer chooses a random $m \times m$ matrix $\boldsymbol{F}$ with entries in $\mathbb{F}$, subject[a] to $\pi_k(\boldsymbol{f}^i) = \boldsymbol{s}_i^\top$, for any $i = 1, \ldots, k$.
2. For any $i = 1, \ldots, N$, the dealer computes $\boldsymbol{h}^i$ and $\boldsymbol{g}_i$ and sends them to P_i
3. Each player P_j sends $\boldsymbol{g}_j \cdot \boldsymbol{m}_i^\top$ to P_i, for $i = 1, \ldots, N$.
4. Each P_i checks, for $j = 1, \ldots, N$, that $\boldsymbol{m}_j \cdot \boldsymbol{h}^i$ equals the value received from P_j. He broadcasts $(\mathsf{accept}, sid, ssid,)$ if all checks are satisfied, otherwise he broadcasts $(\mathsf{reject}, sid, ssid,)$.
5. If all players said $(\mathsf{accept}, sid, ssid,)$, then each P_j stores $\boldsymbol{g}_j[i]$ as his share of $\boldsymbol{s}_i$, for $i = 1, \ldots, k$. Otherwise the protocol aborts.

[a] Recall that we use π_k to denote the projection that outputs the first k coordinates of a vector

Fig. 3. Packed Verifiable Secret-Sharing Scheme

For a column vector $\boldsymbol{v} \in \mathbb{F}^m$, we will say that $\boldsymbol{v}$ *shares* $\boldsymbol{s} \in \mathbb{F}^k$, if $\pi_k(\boldsymbol{v}) = \boldsymbol{s}$ and each honest player P_j holds $\boldsymbol{m}_j \cdot \boldsymbol{v}$. It is clear the the scheme $\boldsymbol{\Pi}_{\mathrm{VSS}}$ is complete, i.e. if the dealer is honest, then all honest players accept and the column vector $\boldsymbol{f}^i$ shares $\boldsymbol{s}_i$, for any $i = 1, \ldots, k$. Moreover, the scheme has the following reconstruction property:

Lemma 1. *Let B be a qualified subset of b honest players and assume that the protocol $\boldsymbol{\Pi}_{\mathrm{VSS}}$ doesn't abort. Then, for all $i = 1, \ldots, k$, the vector $\tilde{\boldsymbol{f}}^i$ (defined by $\tilde{\boldsymbol{f}}^i = \sum_{j=1}^{b} \boldsymbol{r}_{B,i}[j]\boldsymbol{h}^j$) shares $\pi_k(\tilde{\boldsymbol{f}}^i)$. The vectors $\boldsymbol{r}_{B,i}$ are the reconstruction vectors defined in Section 2.3.*

Lemma 1 assures that if the protocol $\mathbf{\Pi}_{\text{VSS}}$ doesn't abort, then, even when the dealer is corrupted, for all $i = 1, \ldots, k$ the info held by a qualified set of honest players at the end of the protocol determine the secret $\boldsymbol{s}_i = \pi_k(\tilde{\boldsymbol{f}}^i)^\top$ and the randomness $\tilde{\boldsymbol{f}}^i$ used by the dealer to share it in such a way that $(\boldsymbol{M} \cdot \tilde{\boldsymbol{f}}^i)[j] = \boldsymbol{m}_j \cdot \tilde{\boldsymbol{f}}^i = \boldsymbol{g}_j[i]$ for any j with P_j honest.

Finally, since $\mathbf{\Pi}_{\text{VSS}}$ shares k secrets in one execution, the privacy property can be stated in an extended form which also guarantees that making public any linear combination of the shared secrets doesn't reveal extra info on the individual secrets.

Lemma 2. *If the dealer in $\mathbf{\Pi}_{\text{VSS}}$ is honest, then for any unqualified set of players A and for any $\lambda_1, \ldots, \lambda_\ell \in \mathbb{F}$, the distribution of $\{\boldsymbol{F} \cdot \boldsymbol{M}_A^\top, \boldsymbol{M}_A \cdot \boldsymbol{F}, \sum_{j=1}^{\ell} \lambda_j \boldsymbol{s}^j\}$ is independent of the secrets held by the dealer.*

5.2 Homomorphic Commitment Scheme

In this section we present our additively homomorphic commitment scheme. The protocol is designed in the $\mathcal{F}_{\text{OT}}^{t-1,t}$-hybrid model using preprocessing and it will be based on the instantiation of the $\mathbf{\Pi}_{\text{VSS}}$ protocol in which the underlying LSSS is the one that is equivalent to our encoding procedure $\mathsf{Enc}_t^{\mathcal{C}}$. The result is a commitment scheme that can be seen as a concrete exemplification of the homomorphic commitment scheme described in [DDGN14]. Note that in this section, for technical reasons, the fixed integer t has to be strictly greater than 2.

Given the $[n, k, d]$ linear error-correcting code $\mathcal{C}$, we have already noted in Remark 1 that computing the vector $\boldsymbol{w} = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m}; \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$ is equivalent to computing the share-vector for $\boldsymbol{m}$ in the LSSS defined by $(k, \boldsymbol{M}_t^{\mathcal{C}})$ for $N = tn$ players. In particular $\boldsymbol{w} = \boldsymbol{M}_t^{\mathcal{C}} \cdot \boldsymbol{f}$ where the vector $\boldsymbol{f}$ is given by $\boldsymbol{f} = (\boldsymbol{m}, \boldsymbol{f}_1, \ldots, \boldsymbol{f}_n)^\top$ with $\boldsymbol{f}_j = (\boldsymbol{v}_1[j], \ldots, \boldsymbol{v}_{t-1}[j])$ for any $j \in [n]$.

The protocol $\mathbf{\Pi}_{\text{HCOM}}$ is presented in Figure 4. In the Setup phase, firstly the same watch-list mechanism of $\mathbf{\Pi}_{\text{COM}}$ is created and after the sender runs $\mathbf{\Pi}_{\text{VSS}}$ on some random messages $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k$ computing the vectors $\boldsymbol{h}^i, \boldsymbol{g}_i$ for all $i = 1, \ldots, N$. In particular P_s computes $\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{r}_i) = (\boldsymbol{g}_1[i], \ldots, \boldsymbol{g}_N[i])^\top$. Thanks to the watch-list mechanism, the receiver sees all the vectors $\boldsymbol{h}^i, \boldsymbol{g}_i$ such that i is in the watch-list set W and therefore it can check the relation $\boldsymbol{m}_j \cdot \boldsymbol{h}^i = \boldsymbol{g}_j \cdot \boldsymbol{m}_i^\top$ for all i, j in W. If all these checks are satisfied, then it follows from the strong reconstruction property of the VSS, that the vectors $\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{r}_i)$ have been properly constructed (i.e. $\Lambda_t(\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{r}_i))$ is a codeword) with overwhelming probability.

Nevertheless, since the set of players $\{P_i \mid i \in W\}$ is unqualified for the LSSS $(k, \boldsymbol{M}_t^{\mathcal{C}})$, the receiver has no info about the vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k$.

In the Online phase, to commit to $\boldsymbol{m} \in \mathbb{F}^k$, the sender takes an unused $\boldsymbol{r}_\eta$ and sends $\boldsymbol{c} = \boldsymbol{m} + \boldsymbol{r}_\eta$ to the sender. The commitment is represented by the pair $(\eta, \boldsymbol{c})$. To open it, the sender reveals $\boldsymbol{m}$ and the randomness used to compute $\boldsymbol{w} = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{r}_\eta)$, thus the receiver can check if the entries he already knows of the encoding of $\boldsymbol{r}_\eta$ match the ones of $\boldsymbol{w}$.

As in the basic protocol, the hiding property follows easily from the privacy of the VSS scheme and the security of the PRG. The binding property, again, follows from the fact that in order to change $\boldsymbol{r}_i$ in $\boldsymbol{r}_i'$ the sender has to change a large amount of entries in $\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{r}_i)$ without knowing which entries the receiver checks. Finally, in this protocol we can implement additions: given a commitment $(\alpha, \boldsymbol{c}_1)$ to $\boldsymbol{m}_1$ and a commitment $(\beta, \boldsymbol{c}_2)$ to $\boldsymbol{m}_2$, both the parties can just compute $\boldsymbol{c}_3 = \boldsymbol{c}_1 + \boldsymbol{c}_2$ and store $((\alpha, \beta), \boldsymbol{c}_3)$ as new commitment. To open $\boldsymbol{c}_3$ to $\boldsymbol{m}_1 + \boldsymbol{m}_2$ the senders sends to P_r the vector $\boldsymbol{m}_1 + \boldsymbol{m}_2$ and the sum of the randomness used in $\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{r}_\alpha)$ an in $\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{r}_\beta)$. While the receiver will check the received randomness as in an usual Open phase but considering the sum of the encodings of $\boldsymbol{r}_\alpha$ and $\boldsymbol{r}_\beta$.

Note that now a commitment will be represented by $(\eta, \boldsymbol{c})$, where η can also be a tuple of indices instead of just one index in $[k] = \{1, \ldots, k\}$. Indeed, if $\boldsymbol{c}$ is the commitment obtained by the sum of ℓ standard commitments (i.e. commitments created in the Commit phase), then $\eta \in [k]^\ell$. For this reason, in order to implement the Addition command in the description of the protocol, we will use the following notation: if $\alpha \in [k]^i$ and $\beta \in [k]^j$, then $\gamma = \alpha \parallel \beta = (\alpha, \beta) \in [k]^{i+j}$.

The protocol $\mathbf{\Pi}_{\mathrm{HCOM}}$ UC-realizes the ideal functionality $\mathcal{F}_{\mathrm{HCOM}}$ in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid model, as stated in the following two propositions. See Appendix A.4 for the proofs.

Proposition 3 (Statistical Binding Property). *Let $G : \{0,1\}^{l'} \to \{0,1\}^{2m}$ be a pseudorandom generator and $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a $[n, k, d]$ error correction code over $\mathbb{F}$. For every static active adversary $\mathcal{A}$ corrupting only P_s in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid world execution of $\mathbf{\Pi}_{\mathrm{HCOM}}$ and for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{HCOM}}, \mathcal{S}, \mathcal{Z}} \approx \mathsf{HYBRID}_{\mathbf{\Pi}_{\mathrm{HCOM}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\mathrm{OT}}^{t-1,t}}$$

where the security parameter is $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Protocol $\mathbf{\Pi}_{\text{HCOM}}$ in the $\mathcal{F}_{\text{OT}}^{t-1,t}$-hybrid model

Let $G : \{0,1\}^{l'} \to \{0,1\}^{2m}$ be a pseudorandom generator, $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a $[n,k,d]$ code over $\mathbb{F}$ and $t \geq 3$ a fixed integer. We recall that $N = tn$, $m = k + (t-1)n$ and the matrix $\boldsymbol{M}_t^{\mathcal{C}}$, whose i-th row is called $\boldsymbol{m}_i$, is defined in Remark 1.
A sender P_s and receiver P_r interact between themselves and with $\mathcal{F}_{\text{OT}}^{t-1,t}$ as follows:

Setup phase:

OT-Setup:

For $i = 1, t+1, 2t+1, \ldots, (n-1)t+1$:

1. P_s samples t strings $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_{i+t-1} \leftarrow \{0,1\}^{l'}$ and sends $(\mathsf{sender}, sid, ssid, (\boldsymbol{x}_i, \ldots, \boldsymbol{x}_{i+t-1}))$ to $\mathcal{F}_{\text{OT}}^{t-1,t}$.
2. P_r samples $\{c_1^i, \ldots, c_{t-1}^i\} \leftarrow \{0, 1, \ldots, t-1\}$ and sends $(\mathsf{receiver}, sid, ssid, c_1^i, \ldots, c_{t-1}^i)$ to $\mathcal{F}_{\text{OT}}^{t-1,t}$.
3. P_r receives $(\mathsf{received}, sid, ssid, \boldsymbol{x}_{i+c_1^i}, \ldots, \boldsymbol{x}_{i+c_{t-1}^i})$ from $\mathcal{F}_{\text{OT}}^{t-1,t}$.

Let $\boldsymbol{Y} \in \mathbf{Mat}_{N\times 2m}$ be the $N \times 2m$ matrix with rows $\boldsymbol{y}_j$'s consisting of the row vectors $G(\boldsymbol{x}_j)$'s for $j = 1, \ldots, N$ and $W = \left\{i + c_1^i, \ldots, i + c_{t-1}^i \mid i = 1, t+1, 2t+1, \ldots, (n-1)t+1\right\}$.

Pre-commitment:

1. Upon receiving $(\mathsf{received}, sid, ssid)$ from $\mathcal{F}_{\text{OT}}^{t-1,t}$, P_s samples $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k \leftarrow \mathbb{F}^k$ and runs $\mathbf{\Pi}_{\text{VSS}}$ $(\boldsymbol{M}_t^{\mathcal{C}})$ using $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k$ as input and constructing the row vectors $\boldsymbol{w}_i = \left(\boldsymbol{g}_i, (\boldsymbol{h}^i)^\top\right) \in \mathbb{F}^{2m}$ for $i = 1, \ldots, N$. Let $\boldsymbol{W} \in \mathbf{Mat}_{N\times 2m}$ be the matrix consisting of the rows $\boldsymbol{w}_i$.
2. P_s computes $\boldsymbol{A} = \boldsymbol{W} + \boldsymbol{Y}$ and sends $(sid, ssid, \boldsymbol{A})$ to P_r. Denote with $\boldsymbol{a}_i$ the i-th row of $\boldsymbol{A}$.
3. P_r computes $\left(\boldsymbol{g}_i, (\boldsymbol{h}^i)^\top\right) = \boldsymbol{a}_i - \boldsymbol{y}_i$ for all $i \in W$ and checks if $\boldsymbol{m}_j \cdot \boldsymbol{h}^i = \boldsymbol{g}_j \cdot \boldsymbol{m}_i^\top$ for all different indices $i, j \in W$. If all the checks are satisfied, then P_r accepts the Setup phase, otherwise it halts.

Commit phase:

1. Upon input $(\mathsf{commit}, sid, ssid, P_s, P_r, \boldsymbol{m})$ for $\boldsymbol{m} \in \mathbb{F}^k$, P_s chooses an unused $\boldsymbol{r}_\eta$ from the Setup phase, computes $\boldsymbol{c} = \boldsymbol{m} + \boldsymbol{r}_\eta$ and sends $(sid, ssid, \eta, \boldsymbol{c})$ to P_r.
2. P_r stores $(sid, ssid, \eta, \boldsymbol{c})$ and outputs $(\mathsf{receipt}, sid, ssid, P_s, P_r)$.

Addition:

If the tuples $(sid, ssid_1, \alpha, \boldsymbol{c}_1)$, $(sid, ssid_2, \beta, \boldsymbol{c}_2)$ were previously sent by P_s and recorded by P_r, then:

1. Upon input $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_s, P_r)$, both the players P_s and P_r define and store $(sid, ssid_3, \gamma, \boldsymbol{c}_3)$ where $\gamma = \alpha \parallel \beta$ and $\boldsymbol{c}_3 = \boldsymbol{c}_1 + \boldsymbol{c}_2$.

Open phase:

If $(sid, ssid, \delta, \boldsymbol{c}')$ was stored and $\delta = (\delta_1, \ldots, \delta_\ell) \in [k]^\ell$, then:

1. Upon input $(\mathsf{reveal}, sid, ssid, P_s, P_r)$ to reveal message $\boldsymbol{m}'$, P_s sends $(sid, ssid, \boldsymbol{m}', \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$ to P_r, where $\boldsymbol{v}_i = \sum_{j=1}^{\ell} \boldsymbol{v}_i^{\delta_j}$ for all $i = 1, \ldots, t-1$ and the vector[a] $\mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{r}_{\delta_j}; \boldsymbol{v}_1^{\delta_j}, \ldots, \boldsymbol{v}_{t-1}^{\delta_j})$ is the column number δ_j in the matrix $\boldsymbol{W}$ (for all $j = 1, \ldots, \ell$).
2. P_r receives $(sid, ssid, \boldsymbol{m}', \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$ and computes $\boldsymbol{w} = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{c}' - \boldsymbol{m}'; \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$. Then, P_r checks if $\boldsymbol{w}[j] = \sum_{i=1}^{\ell} \boldsymbol{g}_j[\delta_i]$ for all the entries $j \in W$. If this check fails P_r rejects the commitment and halts. Otherwise P_r outputs $(\mathsf{reveal}, sid, ssid, P_s, P_r, \boldsymbol{m}')$.

[a] Since the LSSS defined by $(k, \boldsymbol{M}_t^{\mathcal{C}})$ is equivalent to the encoding procedure $\mathsf{Enc}_t^{\mathcal{C}}$, P_s already knows the vectors $\{\boldsymbol{v}_i^{\delta_j}\}_i$ used to encode $\boldsymbol{r}_{\delta_j}$ from the Pre-commitment phase

Fig. 4. Protocol $\mathbf{\Pi}_{\text{HCOM}}$

Also in the protocol $\mathbf{\Pi}_{\mathrm{HCOM}}$ it is possible to implement polynomial many commitments, after having run the OT-Setup phase only once. Indeed, after that the watch-list W has been settled, the sender can always sample new random vectors $\boldsymbol{r}_1^*, \ldots, \boldsymbol{r}_k^* \leftarrow \mathbb{F}^k$ and, together with the receiver, repeat the execution of the Pre-commitment phase on this new input. We have already recalled in Section 4 that it is possible to expand the PRG output in order to have new one-time keys to use in the each execution of the Pre-commitment phase. After that, P_s and P_r can continue the protocol following the instructions in $\mathbf{\Pi}_{\mathrm{HCOM}}$. Moreover, this doesn't create any restriction about the Addition command: we can allow the sum of commitments that use one-time keys coming from different Pre-commitment phases.

Proposition 4 (Computational Hiding Property). *Let $G : \{0,1\}^{l'} \to \{0,1\}^{2m}$ be a pseudorandom generator and $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a $[n,k,d]$ error correction code over $\mathbb{F}$. For every static active adversary $\mathcal{A}$ corrupting only P_r in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid world execution of $\mathbf{\Pi}_{\mathrm{HCOM}}$ and for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{HCOM}},\mathcal{S},\mathcal{Z}} \approx \mathsf{HYBRID}^{\mathcal{F}_{\mathrm{OT}}^{t-1,t}}_{\mathbf{\Pi}_{\mathrm{HCOM}},\mathcal{A},\mathcal{Z}}$$

where the security parameter is $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

6 Complexity and Concrete Efficiency

In this section we discuss the computational and communication complexities of the commitment schemes proposed in Sections 4 and 5. We also estimate concrete parameters and compare the efficiency of our schemes with previous works.

6.1 Complexity

The commitment scheme presented by Damgård *et al.* in [DDGN14] suffered from a quadratic computational overhead in order to achieve optimal communication overhead. This issue stems from the fact that their scheme requires an underlying LSSS that operates over constant size fields [CDP12] whose sharing operations consist in matrix multiplications. Our homomorphic scheme circumvents that by constructing the VSS scheme from a linear error correcting code with linear-time encoding where one can compute shares by computing encodings.

The core component of both commitment schemes is the coding scheme $\mathsf{Enc}_t^{\mathcal{C}}$. This construction can be seen both as an error correcting code (ECC) and a linear secret secret sharing scheme for a specific access structure. $\mathsf{Enc}_t^{\mathcal{C}}$ can be built from *any* linear error correcting code, differently from previous results, which require codes whose dual codes have high minimum distance in order to construct LSSS. This fundamental difference in construction allows us to obtain a coding scheme $\mathsf{Enc}_t^{\mathcal{C}}$ (and consequently a LSSS) that runs in linear time on the input length from any linear-time encodable error correcting code. There exist constructions of linear-time encodable codes with constant rate and good (i.e., linear in the codeword length) minimum distance, see [GI01,GI02,GI03,GI05,Spi96]. However, these may even be more sophisticated than what we need since all we require about the minimum distance is that it is at least $2\tau + 1$, where τ is the security parameter.

The encoding and decoding procedures of $\mathsf{Enc}_t^{\mathcal{C}}$ inherit the complexity of the underlying code. Notice that in our constructions we only utilize the encoding procedure of $\mathsf{Enc}_t^{\mathcal{C}}$, since sharing and verifying share consistency in the VSS scheme of Figure 3 can be seen as encoding. Hence, our constructions can even take advantage of recent advances in linear-time encodable codes [DI14].

Combining a linear-time encoding procedure $\mathsf{Enc}_t^{\mathcal{C}}$ with a PRG where we pay only a constant number of elementary bit operations per output bit (see, *e.g.*, [VZ12]), we obtain UC-secure commitments with optimal computational complexity. Notice that the setup phase (where OTs are needed) is only run once, allowing for an arbitrary number of posterior commitments. Thus, the cost of this phase is amortized over the number of commitments. Communication complexity is also linear in the message length if $\mathcal{C}$ has constant rate.

6.2 Concrete Parameters and Efficiency

Even though our schemes can achieve optimal asymptotic computational and communication complexities, we are also interested in obtaining highly efficient concrete instantiations. As an example, we estimate parameters for a concrete instantiation of our schemes with message length $k = 256$ bits and statistical security parameter $\tau = 60$.

Building Blocks: The basic building blocks of our commitment scheme are the coding scheme $\mathsf{Enc}_t^{\mathcal{C}}$, a PRG and a UC-secure OT protocol. We select the following constructions of these building blocks for our concrete instances:

- **OT:** The UC-secure protocol presented in [PVW08]. This protocol is round optimal and requires communicating 6 group elements and computing 11 exponentiations per transfer.
- **PRG:** AES in counter mode, using the IV as a PRG seed. AES implementations are readily available in modern hardware (*e.g.* Intel's AES-NI) making the cost of this PRG negligible.
- $\mathsf{Enc}_t^{\mathcal{C}}$: For the basic scheme of Figure 2 we will need a $\mathsf{Enc}_2^{\mathcal{C}}$ coding scheme, while for the additively homomorphic scheme of Figure 4 we need a $\mathsf{Enc}_3^{\mathcal{C}}$ coding scheme. Both schemes are constructed using a binary $[796, 256, \geq 121]$ BCH code (see, *e.g.*, [MS78]) [2] as $\mathcal{C}$ according to the generic construction of Section 3. This code has parameters $k = 256$, $n = 796$ and $d \geq 121$, which corresponds to $\tau = 60$. We obtain $\mathsf{Enc}_2^{\mathcal{C}} : \mathbb{F}^{256} \rightarrow \mathbb{F}^{1592}$ and $\mathsf{Enc}_3^{\mathcal{C}} : \mathbb{F}^{256} \rightarrow \mathbb{F}^{2388}$. Even though this code doesn't have linear encoding complexity, it was chosen because it is readily available in the Linux Kernel and it achieves good concrete performance.

Concrete Parameters: Table 1 presents the communication, round and computational complexities in terms of the parameters $[n, k, d]$ of the code $\mathcal{C}$ used to instantiate the encoding scheme $\mathsf{Enc}_t^{\mathcal{C}}$. The commitment message length is k and the statistical security parameter follows from code $\mathcal{C}$'s minimum distance d following the relation $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Notice that the homomorphic scheme of Figure 4 requires a pre-commitment phase after which it is possible to execute k commitments to messages of length k. Hence, the communication and computational complexities of an individual commitment are computed by dividing the total cost of the pre-commitment phase by the number of commitments that can be executed after this phase[3]. In practice the whole pre-commitment must be run before additively homomorphic commitments can be executed. Nevertheless, the pre-commitment phase can be preprocessed, since it is independent of the actual messages.

[2] More precisely, the $[796, 256, \geq 121]$ code is actually obtained by shortening a BCH-code with parameters $[1023, 483, \geq 121]$. This code was in turn selected by first fixing the message size $k = 256$, the statistical security parameter $\tau = 60$ and the minimum distance $d \geq 2\tau + 1 \geq 121$, then using MAGMA to compute concrete code parameters that fit these constraints.

[3] Computing and checking the consistency of shares under the VSS scheme $\mathbf{\Pi}_{\mathrm{VSS}}$ $(\boldsymbol{M}_3^{\mathcal{C}})$ used for the additively homomorphic scheme of Figure 4 can be seen as computing a number of encodings under $\mathsf{Enc}_3^{\mathcal{C}}$.

The setup phase for both schemes requires n executions of a $t-1$-out-of-t OT. Its cost in terms of exponentiations/encodings depends on the OT protocol used.

Scheme	Communication Complexity (in field elements)			Round Complexity		Computational Complexity		
	Commit	**Open**	**Total**	**Commit**	**Open**	**Commit**	**Open**	**Total**
Fig. 4 (homomorphic)	$\frac{2mnt}{k}+k$	m	$\frac{2mnt}{k}+k+m$	1	1	$\frac{4n(t-1)}{k}+2$ Enc.	1 Enc.	$\frac{4n(t-1)}{k}+3$ Enc.
Fig. 2 (basic)	nt	m	$m+nt$	1	1	1 Enc.	1 Enc.	2 Enc.

Table 1. Concrete efficiency in terms of coding scheme $\mathsf{Enc}_t^{\mathcal{C}}$ parameter t and code $\mathcal{C}$ parameters dimension k, and length n. Message length is k and statistical security follows from the relation $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$, depending on code $\mathcal{C}$'s minimum distance d. Enc. stands for encodings and $m = k + n(t-1)$.

Preprocessing: Both the basic scheme of Figure 2 and the homomorphic scheme of Figure 4 can benefit from preprocessing. In this model, the commitment phase is preprocessed before the messages are known in a so called *offline phase*. Later on, in the *online phase*, the sender can commit to its actual messages virtually for free. Using this trick, the sender can pre-compute a number of commitments before they are actually needed during his idle time, dramatically speeding up the online phase where he receives his actual inputs. This strategy is particularly fit for scenarios where a large number of commitments are known to be needed at some point, such as cut-and-choose protocols.

Notice that, in pre-commitment phase of the additively homomorphic scheme of Figure 4, the sender uses the VSS scheme to share several random strings of the same size as the messages and sends a fraction of the resulting shares to the receiver for verification. Later on, in the online phase, the sender can simply encrypt its actual messages using this random strings as one-time pads. A similar trick can be applied to the basic scheme in Figure 2 by using the regular commit phase steps to pre-commit to a series of random strings of the same size as the actual messages before they are known.

Table 2 presents the communication and computational complexities of our schemes in the preprocessing model in terms of the parameters

$[n, k, d]$ of the code $\mathcal{C}$ used to instantiate the encoding scheme $\mathsf{Enc}_t^{\mathcal{C}}$. The commitment message length is k and the statistical security parameter follows from code $\mathcal{C}$'s minimum distance d following the relation $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$. Notice that pre-commitments are run as offline phase.

Scheme	Communication Complexity (in field elements)			Round Complexity			Computational Complexity		
	Offline	**Commit**	**Open**	**Offline**	**Commit**	**Open**	**Offline**	**Commit**	**Open**
Fig. 4 (homomorphic)	$\frac{2mnt}{k}$	k	m	1	1	1	$\frac{4n(t-1)}{k} + 2$ Enc.	0 Enc.	1 Enc.
Fig. 2 (basic)	nt	k	m	1	1	1	1 Enc.	0 Enc.	1 Enc.

Table 2. Preprocessing model concrete efficiency in terms of coding scheme $\mathsf{Enc}_t^{\mathcal{C}}$ parameter t and code $\mathcal{C}$ parameters dimension k, and length n. Message length is k and statistical security follows from the relation $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$, depending on code $\mathcal{C}$'s minimum distance d. Enc. stands for encodings and $m = k + n(t-1)$.

Evaluating Efficiency: Previous efficiency comparisons between UC-secure commitment schemes have been based on the number of exponentiations required by each scheme. This choice of comparison parameters is justified by the fact that this is usually the most costly operation that dominates the concrete execution time of such schemes. However, apart from the setup phase involving OTs, our protocols require no exponentiations at all. After the setup phase of our protocols, the most expensive operation is the encoding procedure of the $\mathsf{Enc}_t^{\mathcal{C}}$ coding scheme (the other operation required is addition).

We compare the efficiency of our schemes with the most efficient previous works [BCPV13,Lin11] by estimating the execution time of the encoding procedure of the BCH code and comparing that to the execution time of exponentiations on the same platform. While the encoding scheme of the ECC and the PRG are used proportionally to the number of commitments one wishes to make and open, the OT protocol is only used for a fixed number of times during the setup phase. Hence, it is interesting to estimate the concrete efficiency of the setup phase separately from the other steps of the protocols, since the cost of running the OT protocol is amortized over the number of commitments.

The concrete computational, round and communication complexities for our schemes when instantiated using the previously described building blocks are presented in Table 3. In this case we consider message length $k = 256$ and statistical security parameter $\tau = 60$, using the $[796, 256, \geq 121]$ BCH code as the building block for $\mathsf{Enc}_2^{\mathcal{C}}$ and $\mathsf{Enc}_3^{\mathcal{C}}$.

Scheme	Communication Complexity (in bits)			Round Complexity		Computational Complexity		
	Commit	**Open**	**Total**	**Commit**	**Open**	**Commit**	**Open**	**Total**
[BCPV13] (Fig. 6)	1024	2048	3072	1	5	10 Exp.	12 Exp.	22 Exp.
[Lin11] (Protocol 2)	1024	2560	3584	1	3	5 Exp.	$18\frac{1}{3}$ Exp.	$23\frac{1}{3}$ Exp.
Fig. 4 (homomorphic, $t = 3$)	34733	1848	36580	1	1	27 Enc.	1 Enc.	28 Enc.
Fig. 2 (basic, $t = 2$)	1592	1052	2644	1	1	1 Enc.	1 Enc.	2 Enc.

Table 3. Concrete efficiency with message length $k = 256$ bits, statistical security parameter $\tau = 60$ and 128-bit computational security (for the schemes of [BCPV13,Lin11]). Exp. and Enc. stand for exponentiations and encodings, respectively.

The execution time of an elliptic curve "exponentiations" over a field of size 256 bits offering 128-bit security is evaluated through an implementation in SCAPI 2.3 [EFLL12] using an underlying curve implementation provided by OpenSSL 1.1.0. The execution time of the encoding procedure of the $[796, 256, \geq 121]$ BCH code is evaluated using the implementation present in the Linux kernel. The platform used for estimating the running time of these operations is based on a Intel(R) Core(TM) i5-2400 CPU at 3.10 GHz with 4 GB of RAM running a Linux Kernel version 3.13.0.

Our experiments showed that the elliptic curve "exponentiations" take an average of 375 μs while the encodings take an average of 0.75 μs on the same platform. Hence, in this scenario, computing one encoding is on average 500 times faster than computing one exponentiation on the same platform. These data show that our basic commitment scheme is 5500 times more computationally efficient than the scheme of [BCPV13], also achieving 14% lower communication complexity. On the other hand, our additively homomorphic commitment scheme is 392 times faster than the scheme of [BCPV13], though its communication complexity is 12 times higher.

The Random Oracle Model [BR93] has historically been used to construct cryptographic schemes with very high efficiency. Surprisingly, our scheme achieves amortised concrete efficiency comparable to previous universally composable schemes based on the ROM [HM04,DSW08] even though it is constructed in the plain model. The average execution time of a SHA-256 hash function in our evaluation platform is of $0.63\mu s$ for the fastest implementation (BouncyCastle) available on SCAPI 2.3, while the OpenSSL implementation runs in $0.835\mu s$. The protocol introduced in [HM04] requires four evaluations of the ROM, which translates into a total execution time 1.68 times higher than of our basic scheme if SHA-256 is used to instantiate the ROM.

Implementing the setup phase required by our basic scheme in Figure 2 requires $n = 796$ executions of a 1-out-of-2 OT, yielding a cost of 8756 exponentiations. With the above timings and considering the OT protocol of [PVW08], the computational complexity of this scheme is lower when at least 398 commitments are computed, and gets increasingly better as the number of commitments increases. However, 4776 of these exponentiations can be precomputed independently of the messages since it is enough for the receiver to get random messages, lowering the online cost to 3980 exponentiations (*i.e.* the cost of 180 commitments. The additively homomorphic scheme in Figure 4 requires $n = 796$ executions of a 2-out-of-3 OT, yielding a higher cost in terms of exponentiations in the setup phase.

References

[BCPV13] Olivier Blazy, Celine Chevalier, David Pointcheval, and Damien Vergnaud. Analysis and improvement of Lindell's UC-secure commitment schemes. In Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 534–551. Springer Berlin Heidelberg, 2013.

[BCR86] G. Brassard, Claude Crepeau, and J.-M. Robert. Information theoretic reductions among disclosure problems. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 168–173, Oct 1986.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS* [DBL01], pages 136–145.

[CDD+14] Ignacio Cascudo, Ivan Damgård, Bernardo David, Irene Giacomelli, Jesper Buus Nielsen, and Roberto Trifiletti. Additively homomorphic UC commitments with optimal amortized overhead. Cryptology ePrint Archive, Report 2014/829, 2014. Full version of PKC 2015 paper.

[CDP12] Ronald Cramer, Ivan Damgård, and Valerio Pastro. On the amortized complexity of zero knowledge protocols for multiplicative relations. In Adam Smith, editor, *ICITS*, volume 7412 of *Lecture Notes in Computer Science*, pages 62–79. Springer, 2012.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[DBL01] *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001.

[DDGN14] Ivan Damgård, Bernardo David, Irene Giacomelli, and Jesper Buus Nielsen. Compact VSS and efficient homomorphic UC commitments. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 213–232. Springer Berlin Heidelberg, 2014.

[DG03] Ivan Damgård and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. In Larmore and Goemans [LG03], pages 426–437.

[DI14] Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 169–182. ACM, 2014.

[DNO10] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. On the necessary and sufficient assumptions for UC computation. In *Theory of Cryptography*, pages 109–127. Springer, 2010.

[DSW08] Yevgeniy Dodis, Victor Shoup, and Shabsi Walfish. Efficient constructions of composable commitments and zero-knowledge proofs. In *Proceedings of the 28th Annual Conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, pages 515–535, Berlin, Heidelberg, 2008. Springer-Verlag.

[EFLL12] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. Scapi: The secure computation application programming interface. Cryptology ePrint Archive, Report 2012/629, 2012. `http://eprint.iacr.org/`.

[GI01] Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA* [DBL01], pages 658–667.

[GI02] Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 812–821. ACM, 2002.

[GI03] Venkatesan Guruswami and Piotr Indyk. Linear time encodable and list decodable codes. In Larmore and Goemans [LG03], pages 126–135.

[GI05] Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005.

[GIKW14] Juan A Garay, Yuval Ishai, Ranjit Kumaresan, and Hoeteck Wee. On the complexity of UC commitments. In *Advances in Cryptology–EUROCRYPT 2014*, pages 677–694. Springer, 2014.

[HM04] Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 58–76. Springer, 2004.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, pages 294–314, 2009.

[LG03] Lawrence L. Larmore and Michel X. Goemans, editors. *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*. ACM, 2003.

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 446–466. Springer, 2011.

[MS78] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, 2nd edition, 1978.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In Michael Wiener, editor, *Advances in Cryptology CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 573–590. Springer Berlin Heidelberg, 1999.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer Berlin Heidelberg, 2008.

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

[VZ12] Salil Vadhan and Colin Jia Zheng. Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In *Proceedings of the 44th symposium on Theory of Computing*, pages 817–836. ACM, 2012.

A Appendix

A.1 Coding Scheme

The following figure illustrates the inner workings of the coding scheme $\mathsf{Enc}_t^{\mathcal{C}}$ introduced in Section 3.

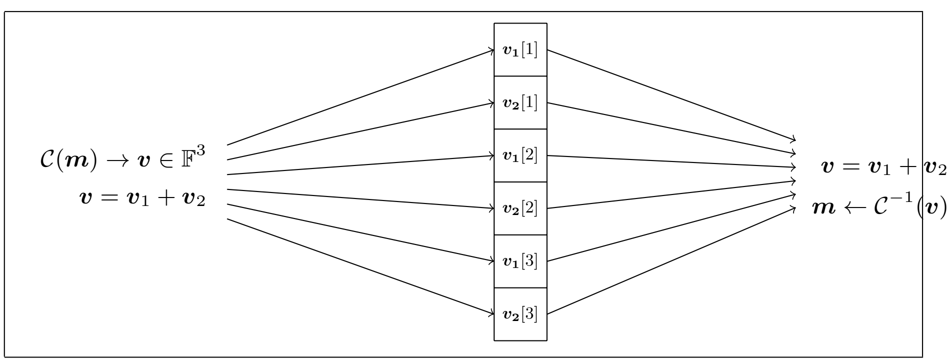

Fig. 5. Coding scheme illustration for $t = 2$, $n = 3$

A.2 Universal Composability

The results presented in this paper are proven secure in the Universal Composability (UC) framework introduced by Canetti in [Can01]. In this framework, protocol security is analyzed under the real-world/ideal-world paradigm, *i.e.* by comparing the real world execution of a protocol with an ideal world interaction with the primitive that it implements. The model has a *composition theorem*, that basically states that UC secure protocols can be arbitrarily composed with each other without any security compromises. This desirable property not only allows UC secure protocols to effectively serve as building blocks for complex applications but also guarantees security in practical environments where several protocols (or individual instances of protocols) are executed in parallel, such as the Internet.

In the UC framework, the entities involved in both the real and ideal world executions are modeled as probabilistic polynomial-time Interactive Turing Machines (ITM) that receive and deliver messages through their input and output tapes, respectively. In the ideal world execution, dummy parties (possibly controlled by an ideal adversary $\mathcal{S}$ referred to as the *simulator*) interact directly with the ideal functionality $\mathcal{F}$, which works as a trusted third party that computes the desired primitive. In the real world execution, several parties (possibly corrupted by a real world adversary $\mathcal{A}$) interact with each other by means of a protocol π that realizes the ideal functionality. The real and ideal executions are controlled by the *environment* $\mathcal{Z}$, an entity that delivers inputs and reads the outputs of the individual parties, the adversary $\mathcal{A}$ and the simulator $\mathcal{S}$. After a real or ideal execution, $\mathcal{Z}$ outputs a bit, which is considered as the output of the execution. The rationale behind this framework lies in showing that the environment $\mathcal{Z}$ (that represents all the things that happen outside of the protocol execution) is not able to efficiently distinguish between the real and ideal executions, thus implying that the real world protocol is as secure as the ideal functionality.

We denote by $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \bar{r})$ the output of the environment $\mathcal{Z}$ in the real-world execution of protocol π between n parties with an adversary $\mathcal{A}$ under security parameter κ, input z and randomness $\bar{r} = (r_{\mathcal{Z}}, r_{\mathcal{A}}, r_{P_1}, \ldots, r_{P_n})$, where $(z, r_{\mathcal{Z}})$, $r_{\mathcal{A}}$ and r_{P_i} are respectively related to $\mathcal{Z}$, $\mathcal{A}$ and party i. Analogously, we denote by $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \bar{r})$ the output of the environment in the ideal interaction between the simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}$ under security parameter κ, input z and randomness $\bar{r} = (r_{\mathcal{Z}}, r_{\mathcal{S}}, r_{\mathcal{F}})$, where $(z, r_{\mathcal{Z}})$, $r_{\mathcal{S}}$ and $r_{\mathcal{F}}$ are respectively related to $\mathcal{Z}$, $\mathcal{S}$ and $\mathcal{F}$. The real world execution and the ideal executions are respectively represented by the ensembles $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}} = \{\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \bar{r})\}_{\kappa\in\mathbb{N}}$ and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} = \{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \bar{r})\}_{\kappa\in\mathbb{N}}$ with $z \in \{0,1\}^*$ and a uniformly chosen $\bar{r}$.

In addition to these two models of computation, the UC framework also considers the $\mathcal{G}$-hybrid world, where the computation proceeds as in the real-world with the additional assumption that the parties have access to an auxiliary ideal functionality $\mathcal{G}$. In this model, honest parties do not communicate with the ideal functionality directly, but instead the adversary delivers all the messages to and from the ideal functionality. We consider the communication channels to be ideally authenticated, so that the adversary may read but not modify these messages. Unlike messages exchanged between parties, which can be read by the adversary, the messages exchanged between parties and the ideal functionality are divided into a *public header* and a *private header*. The public header can be read by the adversary and contains non-sensitive information (such as session identifiers, type of message, sender and receiver). On the other hand, the private header cannot be read by the adversary and contains information such as the parties' private inputs. We denote the ensemble of environment outputs that represents the execution of a protocol π in a $\mathcal{G}$-hybrid model as $\mathsf{HYBRID}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$ (defined analogously to $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$). UC security is then formally defined as:

Definition 3. *A n-party ($n \in \mathbb{N}$) protocol π is said to UC-realize an ideal functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if, for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that, for every environment $\mathcal{Z}$, the following relation holds:*

$$\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathsf{HYBRID}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$$

We say that the protocol is *statistically secure* if the same holds for all $\mathcal{Z}$ with unbounded computing power.

Adversarial Model: In this work we consider security against static adversaries, *i.e.* corruption may only take place *before* the protocols starts execution. We consider active adversaries who may deviate from the protocol in any arbitrary way.

Setup Assumption: It is known that UC commitment protocols (as well as most "interesting" functionalities) cannot be obtained in the plain model [CF01]. In order to overcome this impossibility, UC protocols require a setup assumption, that basically models a resource that is made available to all parties before execution starts. The security of our protocols is proved in the $\mathcal{F}_{\mathrm{OT}}$-hybrid [Can01,CLOS02], where all parties are assumed to have access to an ideal 1-out-of-2 OT functionality (see Figure 8).

Ideal Functionalities: In Section 4, we construct a simple string commitment protocol that UC-realizes the functionality $\mathcal{F}_{\mathrm{COM}}$ as presented in [CLOS02] and recalled here in Figure 6. In Section 5, we extend this simple scheme to allow homomorphic operations over commitments. The extended protocol UC-realizes the functionality $\mathcal{F}_{\mathrm{HCOM}}$ in Figure 7, that basically adds a command for adding two previously stored commitments and an abort command in the Commit Phase to $\mathcal{F}_{\mathrm{COM}}$. The abort is necessary to deal with inconsistent commitments that could be sent by a corrupted party.

In fact, our additively homomorphic commitment protocol is constructed in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid model (*i.e.* assuming access to $(t-1)$-out-of-t OT where $t \geq 2$ is an integer parameter). Notice that $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$ is basically a special case of a k-out-of-n OT where $k = n - 1$, which can be subsequently reduced to the $\mathcal{F}_{\mathrm{OT}}$-hybrid model via standard techniques [Nao91,BCR86,NP99]. We define $\mathcal{F}_{\mathrm{OT}}$ in Figure 8 and $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$ in Figure 9 following the syntax of [CLOS02]. Notice that $\mathcal{F}_{\mathrm{OT}}$ can be efficiently UC-realized by the protocol in [PVW08], which can be used to instantiate the setup phase of our commitment protocols.

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

$\mathcal{F}_{\mathrm{COM}}$ interacts with a sender P_s, a receiver P_r and an adversary $\mathcal{S}$ and it proceeds as follows:

- **Commit Phase**: Upon receiving a message $(\mathsf{commit}, sid, ssid, P_s, P_r, \boldsymbol{m})$ from P_s, where $\boldsymbol{m} \in \{0,1\}^\lambda$, record the tuple $(ssid, P_s, P_r, \boldsymbol{m})$ and send the message $(\mathsf{receipt}, sid, ssid, P_s, P_r)$ to P_r and $\mathcal{S}$. (The lengths of the strings λ is fixed and known to all parties). Ignore any future commit messages with the same *ssid* from P_s to P_r.
- **Open Phase**: Upon receiving a message $(\mathsf{reveal}, sid, ssid)$ from P_s: If a tuple $(ssid, P_s, P_r, \boldsymbol{m})$ was previously recorded, then send the message $(\mathsf{reveal}, sid, ssid, P_s, P_r, \boldsymbol{m})$ to P_r and $\mathcal{S}$. Otherwise, ignore.

Fig. 6. Functionality $\mathcal{F}_{\mathrm{COM}}$

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

$\mathcal{F}_{\text{COM}}$ interacts with a sender P_s, a receiver P_r and an adversary $\mathcal{S}$ and it proceeds as follows:

- **Commit Phase**: Upon receiving a message $(\mathsf{commit}, sid, ssid, P_s, P_r, \boldsymbol{m})$ from P_s, where $\boldsymbol{m} \in \{0,1\}^\lambda$, record the tuple $(ssid, P_s, P_r, \boldsymbol{m})$ and send the message $(\mathsf{receipt}, sid, ssid, P_s, P_r)$ to P_r and $\mathcal{S}$. (The lengths of the strings λ is fixed and known to all parties). Ignore any future commit messages with the same $ssid$ from P_s to P_r. If a message $(\mathsf{abort}, sid, ssid)$ is received from $\mathcal{S}$, the functionality halts.
- **Open Phase**: Upon receiving a message $(\mathsf{reveal}, sid, ssid)$ from P_s: If a tuple $(ssid, P_s, P_r, \boldsymbol{m})$ was previously recorded, then send the message $(\mathsf{reveal}, sid, ssid, P_s, P_r, \boldsymbol{m})$ to P_r and $\mathcal{S}$. Otherwise, ignore.
- **Addition**: Upon receiving a message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_s, P_r)$ from P_s: If tuples $(ssid_1, P_s, P_r, \boldsymbol{m}_1)$, $(ssid_2, P_s, P_r, \boldsymbol{m}_2)$ were previously recorded and $ssid_3$ is unused, record $(ssid_3, P_s, P_r, \boldsymbol{m}_1 + \boldsymbol{m}_2)$ and send the message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_s, P_r, \mathsf{success})$ to P_s, P_r and $\mathcal{S}$.

Fig. 7. Functionality $\mathcal{F}_{\text{HCOM}}$

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

$\mathcal{F}_{\text{OT}}$ interacts with a sender P_s, a receiver P_r and an adversary $\mathcal{S}$, and it proceeds as follows:

- Upon receiving a message $(\mathsf{sender}, sid, ssid, \boldsymbol{x}_0, \boldsymbol{x}_1)$ from P_s, where each $\boldsymbol{x}_i \in \{0,1\}^\lambda$, store the tuple $(ssid, \boldsymbol{x}_0, \boldsymbol{x}_1)$ (The lengths of the strings λ is fixed and known to all parties). Ignore further messages from P_s to P_r with the same $ssid$.
- Upon receiving a message $(\mathsf{receiver}, sid, ssid, c)$ from P_r, where $c \in \{0,1\}$, check if a tuple $(ssid, \boldsymbol{x}_0, \boldsymbol{x}_1)$ was recorded. If yes, send $(\mathsf{received}, sid, ssid, \boldsymbol{x}_c)$ to P_r and $(\mathsf{received}, sid, ssid)$ to P_s and halt. If not, send nothing to P_r (but continue running).

Fig. 8. Functionality $\mathcal{F}_{\text{OT}}$

A.3 UC security for $\boldsymbol{\Pi}_{\text{COM}}$

When both parties are honest, the protocol $\boldsymbol{\Pi}_{\text{COM}}$ is trivially correct. Let $\mathcal{A}$ be a static active adversary that interacts with the sender P_s and the receiver P_r running the protocol $\boldsymbol{\Pi}_{\text{COM}}$ in the $\mathcal{F}_{\text{OT}}^{t-1,t}$-hybrid model. Recalling the notation in Section A.2, we will prove that $\boldsymbol{\Pi}_{\text{COM}}$ UC-realizes the functionality $\mathcal{F}_{\text{COM}}$ by showing a simulator $\mathcal{S}$ that has access to a copy of $\mathcal{F}_{\text{COM}}$ and then arguing that no environment can distinguish with non-negligible probability between its interaction with $\mathcal{S}$ and $\mathcal{F}_{\text{COM}}$ and its interaction with $\mathcal{A}$ and the real parties.

Functionality $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$

$\mathcal{F}_{\mathrm{OT}}^{t-1,t}$ interacts with a sender P_s, a receiver P_r and an adversary $\mathcal{S}$, and it proceeds as follows:

– Upon receiving a message $(\mathsf{sender}, sid, ssid, \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{t-1})$ from P_s, where each $\boldsymbol{x}_i \in \{0,1\}^\lambda$, store the tuple $(ssid, \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{t-1})$.(The lengths of the strings λ is fixed and known to all parties). Ignore further messages from P_s to P_r with the same $ssid$.

– Upon receiving a message $(\mathsf{receiver}, sid, ssid, c_1, \ldots, c_{t-1})$ from P_r, where $c_i \in \{0,1,\ldots,t-1\}$, check if a tuple $(ssid, \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{t-1})$ was recorded. If yes, send $(\mathsf{received}, sid, ssid, \boldsymbol{x}_{c_1}, \ldots, \boldsymbol{x}_{c_{t-1}})$ to P_r and $(\mathsf{received}, sid, ssid)$ to P_s and halt. If not, send nothing to P_r (but continue running).

Fig. 9. Functionality $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$

For the sake of simplicity we analyze separately the cases when only the sender P_s is corrupted and when only the receiver P_r is corrupted.

Proposition 1 (Statistical Binding Property) *Let $G : \{0,1\}^{l'} \to \{0,1\}^l$ be a pseudorandom generator and $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a $[n,k,d]$ error correction code over $\mathbb{F}$. For every static active adversary $\mathcal{A}$ corrupting only P_s in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid execution of $\mathbf{\Pi}_{\mathrm{COM}}$ and for every environment[4] $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{COM}},\mathcal{S},\mathcal{Z}} \approx \mathsf{HYBRID}_{\mathbf{\Pi}_{\mathrm{COM}},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\mathrm{OT}}^{t-1,t}}$$

where the security parameter is $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Proof. The simulator $\mathcal{S}$ proceeds as follows: $\mathcal{S}$ internally runs a copy of $\mathcal{A}$ and delivers to it every input received from $\mathcal{Z}$. Likewise, every output from the internal copy of $\mathcal{A}$ is delivered to $\mathcal{Z}$. After the activation, $\mathcal{S}$ proceeds with the following steps:

1. Simulating the Setup phase: $\mathcal{S}$ receives $(\mathsf{sender}, sid, ssid, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_{i+t-1})$ from $\mathcal{A}$ and samples $\{\tilde{c}_1^i, \ldots, \tilde{c}_{t-1}^i\} \leftarrow \{0,1,\ldots,t-1\}$ for $i = 1, t+1, 2t+1, \ldots, (n-1)t+1$. As in the protocol, define $W = \{i + \tilde{c}_1^i, \ldots, i + \tilde{c}_{t-1}^i \mid i = 1, t+1, 2t+1, \ldots, (n-1)t+1\}$ and $\boldsymbol{Y}$ the $tn \times l$ matrix with rows consisting of the row vectors $G(\boldsymbol{x}_j)$.
2. Simulating the Commit phase: $\mathcal{S}$ receives $(sid, ssid, \eta, \boldsymbol{c})$ from $\mathcal{A}$ and computes $\tilde{\boldsymbol{w}} = \boldsymbol{c} - \boldsymbol{y}^\eta$, where $\boldsymbol{y}^\eta$ is the column number η of the

[4] Note that in the proof of Proposition 1 the requirement for the environment to be polynomial-time is not necessary. Indeed the proof holds for any environment that interacts with each system only a polynomial number of times.

matrix $\boldsymbol{Y}$. Then, $\mathcal{S}$ runs the decoding procedure $\mathsf{Dec}_t^{\mathcal{C}}(\tilde{\boldsymbol{w}})$. If $\mathsf{Dec}_t^{\mathcal{C}}(\tilde{\boldsymbol{w}})$ outputs $\perp$, $\mathcal{S}$ samples a random message $\tilde{\boldsymbol{m}} \leftarrow \mathbb{F}^k$; otherwise it sets $\tilde{\boldsymbol{m}} = \mathsf{Dec}_t^{\mathcal{C}}(\tilde{\boldsymbol{w}})$. $\mathcal{S}$ sends $(\mathsf{commit}, sid, ssid, P_s, P_r, \tilde{\boldsymbol{m}})$ to $\mathcal{F}_{\mathrm{COM}}$.
3. Simulating the Open phase: $\mathcal{S}$ receives $(sid, ssid, \boldsymbol{m}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$ from $\mathcal{A}$ and computes the vector $\boldsymbol{w} = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m}; \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$. If $\boldsymbol{w}[j] = \tilde{\boldsymbol{w}}[j]$ for all $j \in W$, then $\mathcal{S}$ outputs $(\mathsf{reveal}, sid, ssid, P_s, P_r)$ to $\mathcal{F}_{\mathrm{COM}}$. Otherwise $\mathcal{S}$ rejects the commitment and halts.

The simulator always behaves like an honest receiver P_r interacting with $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$ in the hybrid model execution of the protocol. Thus, if $\boldsymbol{m} = \tilde{\boldsymbol{m}}$, the distribution of the messages exchanged with $\mathcal{A}$ in the simulation is exactly the same as in the execution of $\boldsymbol{\Pi}_{\mathrm{COM}}$ and $\mathcal{Z}$ can not distinguish between them.

If $\boldsymbol{m} \neq \tilde{\boldsymbol{m}}$, then in the Open phase $\mathcal{Z}$ can distinguish between the hybrid model execution and the simulated execution when the value $\boldsymbol{m}$ is accepted by $\mathcal{S}$. Indeed in this case only in the hybrid model execution the value $\tilde{\boldsymbol{m}}$ revealed by $\mathcal{F}_{\mathrm{COM}}$ is different from the input value $\boldsymbol{m}$. But we can show that this happens with negligible probability. When $\mathsf{Dec}_t^{\mathcal{C}}(\tilde{\boldsymbol{w}})$ outputs $\perp$ in step 2, then the Hamming distance of $\Lambda_t(\tilde{\boldsymbol{w}})$ from any codeword is strictly greater than τ by definition of $\mathsf{Dec}_t^{\mathcal{C}}$, in particular $\mathrm{d_{Ham}}(\Lambda_t(\boldsymbol{w}), \Lambda_t(\tilde{\boldsymbol{w}})) \geq \tau + 1$. If $\mathsf{Dec}_t^{\mathcal{C}}(\tilde{\boldsymbol{w}})$ doesn't fail, then $C(\tilde{\boldsymbol{m}}) + \boldsymbol{e} = \Lambda_t(\tilde{\boldsymbol{w}})$ (with $\boldsymbol{e}$ vector of weight less or equal than τ) and the Hamming distance of $\Lambda_t(\tilde{\boldsymbol{w}})$ from $\Lambda_t(\boldsymbol{w})$ is strictly greater than τ because

$$\mathrm{d_{Ham}}(\Lambda_t(\boldsymbol{w}), \Lambda_t(\tilde{\boldsymbol{w}})) \geq \mathrm{d_{Ham}}(\Lambda_t(\boldsymbol{w}), \Lambda_t(\tilde{\boldsymbol{w}}) + \boldsymbol{e}) - \mathrm{d_{Ham}}(\boldsymbol{e}, \boldsymbol{0}_{tn}) =$$

$$= \mathrm{d_{Ham}}(\mathcal{C}(\boldsymbol{m}), \mathcal{C}(\tilde{\boldsymbol{m}})) - \mathrm{d_{Ham}}(\boldsymbol{e}, \boldsymbol{0}_{tn}) \geq d - \tau \geq \tau + 1 \; .$$

Thus in both cases, there are at least $\tau + 1$ groups of consecutive entries in $\boldsymbol{w} - \tilde{\boldsymbol{w}}$ in which at least one entry is not zero and therefore the condition $\boldsymbol{w}[j] = \tilde{\boldsymbol{w}}[j]$ for all $j \in W$ for a random W holds with probability equal or less than $\left(\frac{1}{t}\right)^{\tau+1}$.

Proposition 2 (Computational Hiding Property) *Let $G : \{0,1\}^{l'} \rightarrow \{0,1\}^{l}$ be a pseudorandom generator and $\mathcal{C} : \mathbb{F}^k \rightarrow \mathbb{F}^n$ be a $[n, k, d]$ error correction code over $\mathbb{F}$. For every static active adversary $\mathcal{A}$ corrupting only P_r in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid model execution of $\boldsymbol{\Pi}_{\mathrm{COM}}$ and for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{COM}}, \mathcal{S}, \mathcal{Z}} \approx \mathsf{HYBRID}_{\boldsymbol{\Pi}_{\mathrm{COM}}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\mathrm{OT}}^{t-1,t}}$$

where the security parameter is $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Proof. As before, the simulator $\mathcal{S}$ internally runs a copy of $\mathcal{A}$ simulating for it a real-world execution of the protocol $\mathbf{\Pi}_{\mathrm{COM}}$. In particular $\mathcal{S}$ delivers all the messages received from $\mathcal{Z}$ to $\mathcal{A}$ and conversely as if they were communicating directly. In this setting $\mathcal{S}$ is described by the following instructions:

1. Simulating the Setup phase: For $i = 1, t+1, 2t+1, \ldots, (n-1)t+1$: $\mathcal{S}$ samples random strings $\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_{i+1}, \ldots, \tilde{\boldsymbol{x}}_{i+t-1} \leftarrow \{0,1\}^{l'}$ and, acting as $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$, upon receiving $(\mathsf{receiver}, sid, ssid, c_1^i, \ldots, c_{t-1}^i)$ from the internal copy of $\mathcal{A}$, sends back to it $(\mathsf{received}, sid, ssid, \tilde{\boldsymbol{x}}_{i+c_1^i}, \ldots \tilde{\boldsymbol{x}}_{i+c_{t-1}^i})$.
   Denote by W the set $\{i+c_1^i, \ldots, i+c_{t-1}^i \mid i = 1, t+1, 2t+1, \ldots, (n-1)t+1\}$ and by $\boldsymbol{Y}$ the matrix with rows consisting of the row vectors $G(\tilde{\boldsymbol{x}}_j)$.
2. Simulating the Commit phase: After receiving $(\mathsf{receipt}, sid, ssid, P_s, P_r)$ from $\mathcal{F}_{\mathrm{COM}}$, $\mathcal{S}$ chooses an index η such that the η-th column $\boldsymbol{y}^\eta$ is unused, samples $\tilde{\boldsymbol{m}}, \tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_{t-1} \leftarrow \mathbb{F}^k$ and computes $\tilde{\boldsymbol{w}} = \mathsf{Enc}_t^{\mathcal{C}}(\tilde{\boldsymbol{m}}; \tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_{t-1})$. Finally, $\mathcal{S}$ sends $(sid, ssid, \eta, \tilde{\boldsymbol{c}})$ to $\mathcal{A}$ where $\tilde{\boldsymbol{c}} \leftarrow \mathbb{F}^{tn}$ such that $\tilde{\boldsymbol{c}}[j] = \tilde{\boldsymbol{w}}[j] + \boldsymbol{y}^\eta[j]$ for all $j \in W$.
3. Simulating the Open phase: Upon input $(\mathsf{reveal}, sid, ssid, P_r, P_s, \boldsymbol{m})$ from $\mathcal{F}_{\mathrm{COM}}$, $\mathcal{S}$ computes $\boldsymbol{v} = \mathcal{C}(\boldsymbol{m})$. Then it sends $(sid, ssid, \boldsymbol{m}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$ to $\mathcal{A}$, where the vectors $\boldsymbol{v}_k$'s are defined in the following way: fix $j \in \{1, 2, \ldots, n\}$ and let be $i = (j-1)t$, if $\{c_1^i, \ldots, c_{t-1}^i\} = \{0, \ldots, t-2\}$ then just define $\boldsymbol{v}_k[j] = \tilde{\boldsymbol{v}}_k[j]$ for all $k = 1, \ldots, t-1$, otherwise let k_j be the element in $\{0, \ldots, t-2\} \setminus \{c_1^i, \ldots, c_{t-1}^i\}$ and define $\boldsymbol{v}_k[j] = \tilde{\boldsymbol{v}}_k[j]$ for $k \neq k_j$ and $\boldsymbol{v}_{k_j}[j] = \boldsymbol{v}[j] - \sum_{k \neq k_j} \boldsymbol{v}_k[j] - \tilde{\boldsymbol{w}}[jt]$.

In the Setup phase and in the Commit phase the simulator $\mathcal{S}$ behaves like an honest sender running the protocol in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid model, except for the fact that it chooses $\tilde{\boldsymbol{m}}$ at random and the vector $\tilde{\boldsymbol{c}}$ at random under the constraint that it is consistent with the watch-list of $\mathcal{A}$. In the hybrid model all the entries of $\boldsymbol{c}$ are of the form $\boldsymbol{w}[j] + \boldsymbol{y}^\eta[j]$, while in the ideal-world some of the entries of $\tilde{\boldsymbol{c}}$ are replaced by uniformly random elements of $\mathbb{F}$. Thus, if the environment was able to distinguish the distribution of $\boldsymbol{c}$ from the one of $\tilde{\boldsymbol{c}}$, then it would break the computational security property of the PRG used. Regarding the choice of $\tilde{\boldsymbol{m}}$, observe that $\mathcal{A}$ knows only $t-1$ shares of each component of the codewords $\mathcal{C}(\tilde{\boldsymbol{m}})$ (shares in the additive LSSS for t players). Thus by the $(t-1)$-privacy property of the additive LSSS and Remark 1, the vector $\tilde{\boldsymbol{m}}$ is perfectly hidden from $\mathcal{A}$. This allows to conclude that the distribution of $\boldsymbol{c}$ in the hybrid model, given the view of the adversary, is the uniform one over $\mathbb{F}^k$.

Finally, in the Open phase, $\mathcal{S}$ uses its knowledge of the watch-list W in order to compute the vectors $\boldsymbol{m}$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1}$ in such a way that they are consistent with the view of $\mathcal{A}$ from the Setup phase. That is, if $\boldsymbol{w} = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{m}; \boldsymbol{v}_1, \dots, \boldsymbol{v}_{t-1})$, then $\boldsymbol{w}[j] = \tilde{\boldsymbol{w}}[j]$ (the vector $\mathcal{A}$ already knows from the Setup phase) for all $j \in W$. This means that the opening values sent by $\mathcal{S}$ have exactly the same distribution as the values sent in the hybrid model. Since again the distribution of the messages exchanged with $\mathcal{A}$ in all the phases in the simulation is exactly the same as in the hybrid model execution of $\mathbf{\Pi}_{\mathrm{COM}}$, $\mathcal{Z}$ can not distinguish between them.

A.4 UC security for $\mathbf{\Pi}_{\mathbf{HCOM}}$

In the following let $\mathcal{A}$ be a static active adversary that interacts with the sender P_s and the receiver P_r running the protocol $\mathbf{\Pi}_{\mathrm{HCOM}}$ in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid model. For the simulators we will construct to prove the security of the protocol $\mathbf{\Pi}_{\mathrm{HCOM}}$ we will always assume that $\mathcal{S}$ invokes a copy of $\mathcal{A}$ running an internally simulated interaction of $\mathcal{A}$ with the environments and the parties and that $\mathcal{S}$ delivers all messages exchanged between $\mathcal{Z}$ and $\mathcal{A}$ as if they were communicating directly.

Proposition 3 (Statistical Binding Property) *Let $G : \{0,1\}^{l'} \to \{0,1\}^{2m}$ be a pseudorandom generator and $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a $[n, k, d]$ error correction code over $\mathbb{F}$. For every static active adversary $\mathcal{A}$ corrupting only P_s in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid world execution of $\mathbf{\Pi}_{\mathrm{HCOM}}$ and for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{HCOM}},\mathcal{S},\mathcal{Z}} \approx \mathsf{HYBRID}_{\mathbf{\Pi}_{\mathrm{HCOM}},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\mathrm{OT}}^{t-1,t}}$$

where the security parameter is $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Proof. The simulator $\mathcal{S}$ is described by the following instructions:

1. Simulating the OT-Setup phase: $\mathcal{S}$ receives $(\mathsf{sender}, sid, ssid, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \dots, \boldsymbol{x}_{i+t-1})$ from $\mathcal{A}$ and samples the values $\{\tilde{c}_1^i, \dots, \tilde{c}_{t-1}^i\} \leftarrow \{0, 1, \dots, t-1\}$ for $i = 1, t+1, 2t+1, \dots, (n-1)t+1$. Define the watch-list set to be $W = \{i + \tilde{c}_1^i, \dots, i + \tilde{c}_{t-1}^i \mid i = 1, t+1, 2t+1, \dots, (n-1)t+1\}$.
2. Simulating the Pre-commitment phase: $\mathcal{S}$ also receives $(sid, ssid, \boldsymbol{A})$ from $\mathcal{A}$ and it computes the matrix $\boldsymbol{W} = \boldsymbol{A} - \boldsymbol{Y}$ where $\boldsymbol{Y}$ is the matrix with rows $G(\boldsymbol{x}_i)$. If $\boldsymbol{w}_i = \left(\boldsymbol{g}_i, (\boldsymbol{h}^i)^\top\right)$ is the i-th row of $\boldsymbol{W}$, then $\mathcal{S}$ checks if $\boldsymbol{m}_j \cdot \boldsymbol{h}^i = \boldsymbol{g}_j \cdot \boldsymbol{m}_i^\top$ for any $i, j \in W$. If all the checks succeed, $\mathcal{S}$ continues; otherwise it sends abort to $\mathcal{F}_{\mathrm{HCOM}}$ and halts.
   If it doesn't abort, $\mathcal{S}$ uses all the vectors $\boldsymbol{w}_i$ to construct a qualified set H of consistent VSS players as follows: initially H contains all players.

Now, if H contains a pair of inconsistent players P_i, P_j, i.e., where we have $\boldsymbol{m}_j \cdot \boldsymbol{h}^i \neq \boldsymbol{g}_j \cdot \boldsymbol{m}_i^\top$, then these two players are deleted from H. We repeat until no further pairs can be deleted. If H has cardinality equal or greater than $tn - \tau$, then H is qualified and $\mathcal{S}$ can compute the set of vectors $\tilde{\boldsymbol{r}}_1, \dots, \tilde{\boldsymbol{r}}_k$ (see Remark 1 and Lemma 1). Otherwise, it sends abort to $\mathcal{F}_{\text{HCOM}}$ and halts.

3. Simulating the Commit phase: $\mathcal{S}$ receives $(sid, ssid, \eta, \boldsymbol{c})$ and computes $\tilde{\boldsymbol{m}} = \boldsymbol{c} - \tilde{\boldsymbol{r}}_\eta$. It then sends $(\mathsf{commit}, sid, ssid, P_s, P_r, \tilde{\boldsymbol{m}})$ to $\mathcal{F}_{\text{HCOM}}$.
4. Simulating the Addition: Assume that $(sid, ssid_1, \alpha, \boldsymbol{c}_1)$ and $(sid, ssid_2, \beta, \boldsymbol{c}_2)$ has been already stored. If $\mathcal{S}$ receives $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_s, P_r)$ from $\mathcal{A}$, then it define and store $(sid, ssid_3, \gamma, \boldsymbol{c}_3)$ where $\gamma = \alpha \parallel \beta$ and $\boldsymbol{c}_3 = \boldsymbol{c}_1 + \boldsymbol{c}_2$.
5. Simulating the Open phase: Assume that $(sid, ssid, \delta, \boldsymbol{c}')$ was stored and $\delta = (\delta_1, \dots, \delta_\ell) \in [k]^\ell$. If $\mathcal{S}$ receives $(sid, ssid, \boldsymbol{m}', \boldsymbol{v}_1, \dots, \boldsymbol{v}_{t-1})$ from $\mathcal{A}$, it computes $\boldsymbol{w} = \mathsf{Enc}_t^{\mathcal{C}}(\boldsymbol{c}' - \boldsymbol{m}'; \boldsymbol{v}_1, \dots, \boldsymbol{v}_{t-1})$ and checks if $\boldsymbol{w}[j] = \sum_{i=1}^{\ell} \boldsymbol{g}_j[\delta_i]$ for all the entries $j \in W$. If this check fails $\mathcal{S}$ rejects the commitment and halts. Otherwise $\mathcal{S}$ outputs $(\mathsf{reveal}, sid, ssid, P_s, P_r)$ to $\mathcal{F}_{\text{HCOM}}$.

Given the previous instructions for $\mathcal{S}$, we will now argue that the environment $\mathcal{Z}$ can not successfully tell whether it is interacting with $\mathcal{A}$ and the parties running $\boldsymbol{\Pi}_{\text{HCOM}}$, or with $\mathcal{S}$ and $\mathcal{F}_{\text{OT}}^{t-1,t}$ in the ideal execution more than negligible probability.

In the Setup phase, if the checks in step 2 don't succeed or if they succeed and $\mathcal{S}$ can extract the vectors $\tilde{\boldsymbol{r}}_1, \dots, \tilde{\boldsymbol{r}}_k$ from his view, then $\mathcal{S}$ acts like a honest receiver P_r. So in this case the distribution of the messages exchanged with $\mathcal{A}$ is the same as in the protocol $\boldsymbol{\Pi}_{\text{HCOM}}$. Instead, if the checks in step 2 succeed but the reconstruction of the vectors $\tilde{\boldsymbol{r}}_1, \dots, \tilde{\boldsymbol{r}}_k$ fails, then the simulator $\mathcal{S}$ has to abort and the environment will see a difference between the two distributions. However we can prove that this last case only happens with negligible probability. In other words we want to upper-bounded the probability of the event: H has cardinality strictly less than $tn - \tau$ and the checks in step 2 are satisfied. And for this, it is enough to upper-bounded the probability that the checks are satisfied, assuming that the complement of H has cardinality greater or equal to $\tau + 1$. It is clear that the probability that the checks in step 2 are satisfied is less or equal the probability that W doesn't contain pairs of indices i, j such that P_i, P_j are both in the complement of H. So we focus our attention on the latter probability that will be called p in the following. Recall the definition of the players' subsets $T_j = \left\{P_{(j-1)t+1}, \dots, P_{jt}\right\}$ for

$j \in [n]$ and consider a pair of players $P_a \in T_i$ and $P_b \in T_j$. If $i \neq j$, then the probability that the pair a, b is in W is $\left(\frac{t-1}{t}\right)^2$. While if $i = j$, then the probability of the same event is equal to $\frac{t-2}{t}$. Since $\left(\frac{t-1}{t}\right)^2 > \frac{t-2}{t}$, the worst case is when the complement of H is formed only by pairs of players P_a, P_b both coming from the same subset T_i. Now, we observe that if in the complement of H there are two pairs P_a, P_b and P_c and P_d coming from the same subset T_i (a, b, c and d all distinct by construction of H), then by definition of W one of the two pairs $\{a, b\}$, $\{c, d\}$ must stay in W, so $p = 0$. Otherwise we can assume that for each i, in the complement of H there is at most one pair from the subset T_i. In this case the probability p is less or equal to $\left(\frac{2}{t}\right)^{\frac{\tau+1}{2}}$.

In the Commit phase and in the Addition phase $\mathcal{S}$ behaves as an honest receiver, so also in these phases the distribution of the messages exchanged with $\mathcal{A}$ is the same as in the protocol $\mathbf{\Pi}_{\mathrm{HCOM}}$. The only case in which $\mathcal{Z}$ sees a discrepancy is in the Open phase when the message $\boldsymbol{m}'$ sent by $\mathcal{A}$ is accepted but it is different from the message $\tilde{\boldsymbol{m}}$ reveled by $\mathcal{F}_{\mathrm{HCOM}}$. Now we will argue that this happens with negligible probability. Note that, if $\mathcal{S}$ doesn't halt during step 2, then for all $\eta = 1, \ldots, k$ it computes $\tilde{\boldsymbol{r}}_\eta$ together with some randomness $\{\tilde{\boldsymbol{v}}_j^\eta\}_{j=1,\ldots,t-1}$ in such a way that if $\tilde{\boldsymbol{w}}^\eta = \mathsf{Enc}_t^{\mathcal{C}}(\tilde{\boldsymbol{r}}_\eta; \tilde{\boldsymbol{v}}_1^\eta, \ldots, \tilde{\boldsymbol{v}}_{t-1}^\eta)$, then $\tilde{\boldsymbol{w}}^\eta[j] = \boldsymbol{g}_j[\eta]$ for any j such that $P_j \in H$ (see Lemma 1). Therefore, $\Lambda_t(\sum_{i=1}^{\ell} \tilde{\boldsymbol{w}}^{\delta_i})$ is a codeword and its distance from the vector $\Lambda_t(\sum_{i=1}^{\ell} \boldsymbol{w}^{\delta_i})$ is less or equal than τ. Recall that $\boldsymbol{w}^\eta = (\boldsymbol{g}_1[\eta], \ldots, \boldsymbol{g}_{nt}[\eta])^\top$ is the η-th column in the matrix $\boldsymbol{W}$. Thus, if $\tilde{\boldsymbol{m}} \neq \boldsymbol{m}'$, it must hold that $\mathrm{d}_{\mathrm{Ham}}\left(\Lambda_t(\sum_{i=1}^{\ell} \boldsymbol{w}^{\delta_i}), \Lambda_t(\boldsymbol{w})\right) > \tau$, which implies that the checks in step 3 are satisfied with negligible probability, less or equal than $\left(\frac{1}{t}\right)^{\tau+1}$.

Also in the protocol $\mathbf{\Pi}_{\mathrm{HCOM}}$ it is possible to implement polynomial many commitments, after having run the OT-Setup phase only once. Indeed, after that the watch-list W has been settled, the sender can always sample new random vectors $\boldsymbol{r}_1^*, \ldots, \boldsymbol{r}_k^* \leftarrow \mathbb{F}^k$ and, together with the receiver, repeat the execution of the Pre-commitment phase on this new input. We have already recalled in Section 4 that it is possible to expand the PRG output in order to have new one-time keys to use in the each execution of the Pre-commitment phase. After that, P_s and P_r can continue the protocol following the instructions in $\mathbf{\Pi}_{\mathrm{HCOM}}$. Moreover, this doesn't create any restriction about the Addition command: we can allow the sum of commitments that use one-time keys coming from different Pre-commitment phases.

Assume that in step 5 of the simulation for corrupted senders, $(\delta, \boldsymbol{c}')$ has been created by adding commitments from different Pre-commitment phases. For the sake of simplicity, assume that $\delta = (\delta_1, \delta_2)$. The two individual commitments $(\delta_1, \boldsymbol{c}_1)$ and $(\delta_2, \boldsymbol{c}_2)$ use one-time keys $\tilde{\boldsymbol{r}}_{\delta_1}$ and $\tilde{\boldsymbol{r}}^*_{\delta_2}$ that are part of the input of two different executions of the $\mathbf{\Pi}_{\mathrm{VSS}}$ protocol. In this case, in step 2 of the simulation we can have two different qualified set H, H^* used by $\mathcal{S}$ to compute $\tilde{\boldsymbol{r}}_{\delta_1}$, $\tilde{\boldsymbol{r}}^*_{\delta_2}$ respectively. But if the cardinality of $H \cap H^*$ is greater or equal to $nt - \tau$, then we can assure again that $\mathrm{d}_{\mathrm{Ham}}(\Lambda_t(\tilde{\boldsymbol{w}}^{\delta_1} + \tilde{\boldsymbol{w}}^{*\delta_2}), \Lambda_t(\boldsymbol{w}^{\delta_1} + \boldsymbol{w}^{*\delta_2})) \leq \tau$ and we can conclude as in the proof of Proposition 3 (note that the watch-list W is the same in the two executions of $\mathbf{\Pi}_{\mathrm{VSS}}$). Now notice that, if the complement of $H \cap H^*$ has cardinality greater than τ, then we can repeat the argument we had for step 2 and say that the probability that P_i and P_j are neither both in the complement of H nor both in the complement of H^* for all i, j in W is negligible. Indeed, in this analysis only the numbers of inconsistent pairs matters, it doesn't count from which execution the inconsistent pair comes from. Therefore we can assure that if the checks in step 2 are satisfied for all the executions of the VSS scheme, then with overwhelming probability $|H \cap H^*| \geq nt - \tau$. This argument can be easily generalized to the sum of three or more commitments (i.e. $\ell \geq 3$).

Proposition 4 (Computational Hiding Property) *Let $G : \{0,1\}^{l'} \to \{0,1\}^{2m}$ be a pseudorandom generator and $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a $[n, k, d]$ error correction code over $\mathbb{F}$. For every static active adversary $\mathcal{A}$ corrupting only P_r in the $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$-hybrid world execution of $\mathbf{\Pi}_{\mathrm{HCOM}}$ and for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{HCOM}}, \mathcal{S}, \mathcal{Z}} \approx \mathsf{HYBRID}^{\mathcal{F}_{\mathrm{OT}}^{t-1,t}}_{\mathbf{\Pi}_{\mathrm{HCOM}}, \mathcal{A}, \mathcal{Z}}$$

where the security parameter is $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Proof. The simulator $\mathcal{S}$ in this case is similar to the one described for proving the hiding property of the protocol $\mathbf{\Pi}_{\mathrm{COM}}$.

1. Simulating the OT-Setup phase: For $i = 1, t+1, 2t+1, \ldots, (n-1)t+1$: $\mathcal{S}$ samples random strings $\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_{i+1}, \ldots, \tilde{\boldsymbol{x}}_{i+t-1} \leftarrow \{0,1\}^{l'}$ and, acting as $\mathcal{F}_{\mathrm{OT}}^{t-1,t}$, upon receiving $(\mathsf{receiver}, sid, ssid, c_1^i, \ldots, c_{t-1}^i)$ from the internal copy of $\mathcal{A}$, sends back to it $(\mathsf{received}, sid, ssid, \tilde{\boldsymbol{x}}_{i+c_1^i}, \ldots \tilde{\boldsymbol{x}}_{i+c_{t-1}^i})$.
Denote by W the set $\{i + c_1^i, \ldots, i + c_{t-1}^i \mid i = 1, t+1, 2t+1, \ldots, (n-1)t+1\}$ and by $\boldsymbol{Y}$ the matrix with rows consisting of the row vectors $G(\tilde{\boldsymbol{x}}_j)$.

2. Simulating the Pre-commitment phase: $\mathcal{S}$ uniformly samples k random strings $\tilde{\boldsymbol{r}}_1, \ldots, \tilde{\boldsymbol{r}}_k \in \mathbb{F}^k$ and runs the protocol $\boldsymbol{\Pi}_{\mathrm{VSS}}$ $(\boldsymbol{M}_t^{\mathcal{C}})$ on them. In this way it constructs the row vectors $\tilde{\boldsymbol{w}}_i = \left((\tilde{\boldsymbol{h}}^i)^\top, \tilde{\boldsymbol{g}}_i\right) \in \mathbb{F}^{2m}$ for $i = 1, \ldots, N$. Let $\tilde{\boldsymbol{W}}$ be the matrix consisting of the rows $\tilde{\boldsymbol{w}}_i$. $\mathcal{S}$ samples a matrix $\boldsymbol{M} \leftarrow \mathbf{Mat}_{tn\times 2m}$ such that for any $j \in W$ the j-th row of $\boldsymbol{M}$ is given by $\tilde{\boldsymbol{w}}_j + G(\tilde{\boldsymbol{x}}_j)$ and sends $(sid, ssid, \boldsymbol{M})$ to $\mathcal{A}$.
3. Simulating the Commit phase: After receiving $(\mathsf{receipt}, sid, ssid, P_s, P_r)$ from $\mathcal{F}_{\mathrm{HCOM}}$, $\mathcal{S}$ chooses an index η such that random string $\tilde{\boldsymbol{r}}_\eta$ is unused, samples a random message $\tilde{\boldsymbol{c}} \in \mathbb{F}^k$ and sends $(sid, ssid, \eta, \tilde{\boldsymbol{c}})$ to $\mathcal{A}$.
4. Simulating the Addition: Assume that $(sid, ssid_1, \alpha, \tilde{\boldsymbol{c}}_1)$ and $(sid, ssid_2, \beta, \tilde{\boldsymbol{c}}_2)$ has been created. If $\mathcal{S}$ receives the message $(\mathsf{add}, sid, ssid_1, ssid_2, ssid_3, P_s, P_r, \mathsf{success})$ from $\mathcal{F}_{\mathrm{HCOM}}$, it stores $(sid, ssid_3, \gamma, \tilde{\boldsymbol{c}}_3)$ where $\gamma = \alpha \| \beta$ and $\tilde{\boldsymbol{c}}_3 = \tilde{\boldsymbol{c}}_1 + \tilde{\boldsymbol{c}}_2$.
5. Simulating the Open phase: Assume that $(sid, ssid, \delta, \tilde{\boldsymbol{c}}')$ was stored and $\delta = (\delta_1, \ldots, \delta_\ell) \in [k]^\ell$. Upon input $(\mathsf{reveal}, sid, ssid, P_r, P_s, \boldsymbol{m}')$ from $\mathcal{F}_{\mathrm{HCOM}}$, $\mathcal{S}$ computes $\boldsymbol{v} = \mathcal{C}(\tilde{\boldsymbol{c}}' - \boldsymbol{m}')$. Note that from step 2, $\mathcal{S}$ knows the vectors $\tilde{\boldsymbol{v}}_1^i, \ldots, \tilde{\boldsymbol{v}}_{t-1}^i$ such that for $i = 1, \ldots, k$ the column i of $\tilde{\boldsymbol{W}}$ satisfies $\tilde{\boldsymbol{w}}^i = \mathsf{Enc}_t^{\mathcal{C}}(\tilde{\boldsymbol{r}}_i; \tilde{\boldsymbol{v}}_1^i, \ldots, \tilde{\boldsymbol{v}}_{t-1}^i)$. Thus, it can compute the vectors $\boldsymbol{v}_h$'s in the following way: fix $j \in \{1, 2, \ldots, n\}$ and let $i = (j-1)t$, if $\{c_1^i, \ldots, c_{t-1}^i\} = \{0, \ldots, t-2\}$ then just define $\boldsymbol{v}_h[j] = \sum_{s=1}^{\ell} \tilde{\boldsymbol{v}}_h^{\delta_s}[j]$ for all $h = 1, \ldots, t-1$, otherwise let h_j be the element in $\{0, \ldots, t-2\} \setminus \{c_1^i, \ldots, c_{t-1}^i\}$ and define $\boldsymbol{v}_h[j] = \sum_{s=1}^{\ell} \tilde{\boldsymbol{v}}_h^{\delta_s}[j]$ for $h \neq h_j$ and $\boldsymbol{v}_{h_j}[j] = \boldsymbol{v}[j] - \sum_{k \neq h_j} \boldsymbol{v}_h[j] - \sum_{s=1}^{\ell} \tilde{\boldsymbol{w}}^{\delta_s}[jt]$. Finally, $\mathcal{S}$ sends $(sid, ssid_3, \boldsymbol{m}', \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$ to $\mathcal{A}$.

Notice that the set of players $\{P_i \mid i \in W\}$ is unqualified for the LSSS $(k, \boldsymbol{M}_t^{\mathcal{C}})$. Thus, thanks to Lemma 2 arguments similar to the ones used in the proof of Proposition 2 show that the distribution of the messages exchanged with $\mathcal{A}$ in the simulation is the same as in the hybrid-world execution of $\boldsymbol{\Pi}_{\mathrm{HCOM}}$. Thus the environment $\mathcal{Z}$ can not distinguish between its interaction with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{HCOM}}$ and its interaction with $\mathcal{A}$ and the real parties.