Functional Commitments for All Functions, with Transparent Setup and from SIS

Leo de Castro[*‡]　　　Chris Peikert[†‡]

February 28, 2023

Abstract

A *functional commitment* scheme enables a user to concisely commit to a function from a specified family, then later concisely and verifiably reveal values of the function at desired inputs. Useful special cases, which have seen applications across cryptography, include vector commitments and polynomial commitments.

To date, functional commitments have been constructed (under falsifiable assumptions) only for functions that are essentially *linear*, with one recent exception that works for arbitrarily complex functions. However, that scheme operates in a strong and non-standard model, requiring an online, trusted authority to generate special keys for any opened function inputs.

In this work, we give the first functional commitment scheme for nonlinear functions—indeed, for *all functions* of any bounded complexity—under a standard setup and a falsifiable assumption. Specifically, the setup is "transparent," requiring only public randomness (and not any trusted entity), and the assumption is the hardness of the standard Short Integer Solution (SIS) lattice problem. Our construction also has other attractive features, including: *stateless updates* via generic composability; excellent *asymptotic efficiency* for the verifier, and also for the committer in important special cases like vector and polynomial commitments, via preprocessing; and *post-quantum security*, since it is based on SIS.

1 Introduction

In a *functional commitment* scheme, a user first commits to a function f from some specified family. Later, the user can *open* the function at one or more desired inputs x_i, generating noninteractive and publicly verifiable *proofs* for the claimed values $y_i = f(x_i)$, which can be checked against the original commitment.[1] In order to be nontrivial, commitments and proofs should be *concise*, i.e., significantly smaller than (i.e., sublinear or better in) the function's representation. The primary security property of interest is *evaluation binding*: a (possibly maliciously generated) commitment should fix some underlying function. That is, it should be infeasible for an attacker to produce a commitment along with valid proofs for two different function values

[*] EECS, Massachusetts Institute of Technology, `ldec@mit.edu`.

[†] Computer Science and Engineering, University of Michigan, `cpeikert@umich.edu`. This material is based upon work supported by NSF Grant No. CCF-2006857 and by DARPA under Agreement No. HR00112020025. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government, the NSF, or DARPA.

[‡] Some of this work was done while at Algorand, Inc.

[1] Some works consider a dual notion, in which the user commits to some data x and then can open various functions f of it. Our present formulation is a natural one for most specific purposes of interest. Moreover, assuming a sufficiently expressive scheme of either form, its dual notion can be achieved by swapping "code" and "data" using a universal function; see Section 4.3 for details.

$y \neq y'$ at a single input x (all chosen by the attacker). In this work we will not focus on other security properties like function hiding or zero-knowledge for proofs, since they are often not needed in applications, and when they are needed, they can usually be added using standard techniques.

The notion of functional commitments was first formally defined in [LRY16] to encompass prior notions for specific kinds of functionalities, like vector commitments [LY10, CF13], polynomial commitments [KZG10, PST13, LRY16], and linear commitments [LRY16].[2] These kinds of constructions have had a wealth of applications across cryptography, including verifiable outsourcing of storage [BGV11], authenticated streaming data structures [PSTY13], updateable zero-knowledge sets and databases [MRK03, Lis05], cryptographic accumulators [BdM93], pseudonymous credentials [KZG10], stateless transaction validation in cryptocurrencies [CPSZ18], verifiable secret sharing [CGMA85], content-extraction signatures [SBZ01], proof-carrying data systems, and succinct noninteractive arguments (of knowledge), also known as SNARGs/SNARKs [BFS20, BDFG21].

An important bonus feature of functional commitments, which is needed for several of the above-cited applications, is *(stateless) updateability*. This means that it is possible to concisely update the commitment and proofs for some function f to ones for a related function f', *without needing to know* f. For example, a user—who may or may not be the original committer—may wish to define $f' = f + \delta$ for some "update" function δ, and distribute corresponding updates to existing proofs. This functionality can enable authentication for "streaming" data structures and the like (see, e.g., [PSTY13]).

Beyond linearity. As noted in [LRY16], there is a simple, generic construction of functional commitments for *arbitrary* functions, which combines an ordinary concise commitment and a SNARG (see also [BNO21]). In turn, SNARGs can be constructed in the random-oracle model, by making a succinct *interactive* argument (e.g., [Kil92, Mic94]) non-interactive via the Fiat–Shamir transform [FS86]. However, in the absence of random oracles, SNARGs are powerful tools whose known constructions are heuristic, since their security cannot be based on any falsifiable assumption via a black-box security reduction [GW11].

Until quite recently, all known functional commitment schemes (without random oracles) from *falsifiable* assumptions were limited to classes of *linearizable* functions, i.e., ones that can be expressed as linear functions of a suitably "preprocessed" input. This includes the recent constructions of [LP20, CFT22, ACL$^+$22] for "semi-sparse" or constant-degree multi-variate polynomials. In particular, a polynomial can be expressed as the (linear) inner product between the vectors of the polynomial's coefficients and the powers of the input; the preprocessing therefore computes these powers.

Recent work [PPS21] overcame this linearizability barrier for the first time. It obtained functional commitments for functions of *any (bounded) complexity* under the standard Short Integer Solution (SIS) assumption, and thereby worst-case lattice assumptions [Ajt96], via techniques from fully homomorphic encryption and commitments [GSW13, GVW15b]. (These homomorphic schemes are not succinct, so they do not immediately yield functional commitments.)

However, the construction from [PPS21] has a major drawback: it operates in a non-standard model that requires an *online trusted authority*. As in other schemes requiring "structured" public parameters, the authority generates such parameters together with a secret "trapdoor" (which can be used to break the scheme's security). But in addition, the authority must remain online to generate *opening keys* for any inputs at which users wish to open their committed functions.[3] So, this model requires a strong trust assumption.

[2] Earlier works [IKO07, BC12] considered similar concepts, but allowing for interaction and private verifier randomness during the commitment and proving phases. Functional commitments require noninteractive commitments and proofs, and public verifiability.

[3] The work of [PPS21] actually uses the dual formulation of functional commitments, where data x is committed and then functions f of the data are opened. However, the construction easily adapts to our present formulation of functional commitments.

1.1 Our Contributions

Our main contribution resolves the main problem left open by [PPS21]: we construct a functional commitment scheme for *all functions* of a-priori bounded complexity, whose setup needs only an *"unstructured" uniformly random string* (and no trusted authority, online or otherwise), and whose security is based on the standard SIS lattice problem. Such a "transparent" setup of the public parameters is very attractive, because it requires only a public source of trustworthy randomness (which can even be heuristically expanded to the needed size using a cryptographic hash function), and because no entity ever knows a trapdoor that could be use to break the scheme's security.

In particular, we obtain the first constructions, with a standard setup, of polynomial commitments and general linear commitments from standard lattice assumptions (or falsifiable post-quantum assumptions more broadly), and of non-linearizable functional commitments under *any* falsifiable assumption.[4] Moreover, our construction has several other attractive features:

- It is *statelessly updateable* in very general ways, via generic composition properties. In particular, a committed function may be updated additively or multiplicatively, or even generically composed with another function, i.e., outputs of committed functions can be post-processed. The updated commitment and proofs are obtained simply by operating on the original ones according to the update function.

- It *efficiently specializes* to particular functionalities of interest, like vector commitments and polynomial commitments. In these settings and others, the public parameters can first be preprocessed, even by an untrusted party. Then, committing to a function and opening it become relatively fast, highly parallel linear operations (i.e., just one matrix-matrix or even matrix-vector multiplication). Moreover, for vector commitments, the sizes of our commitments and proofs asymptotically beat or essentially match those of prior SIS-based constructions [PSTY13, PPS21] (see Table 1), even though our scheme has simpler opening and verification algorithms (after preprocessing).

- The *verifier is efficient and essentially linear*: it simply checks a single n-dimensional (inhomogeneous) SIS relation, i.e., a short integral solution $\mathbf{S}$ to a linear equation $\mathbf{AS} = \mathbf{Y}$, where the matrices $\mathbf{A}, \mathbf{Y}$ are constructed linearly from the public parameters, the commitment, and the claimed input-output pair. This makes verification highly amenable to recent techniques for proving relations of this kind in zero knowledge, and even succinctly (e.g., [BBC$^+$18, BLS19, LNP22, ACL$^+$22]).

- A variant (using the "context hiding" technique of [GVW15b]) has *(statistical) zero knowledge* commitments and proofs, though with a weaker notion called "target" evaluation binding, which requires the commitment to be generated *honestly*. Zero knowledge means that the commitments and proofs reveal nothing more about the committed function than what is revealed by the provided input-output pairs.

[4] We caution that some works on polynomial commitment schemes (e.g., [BDFG21]) consider additional requirements, e.g., that the committed function is indeed a polynomial of some bounded degree, or even that openings are proofs of *knowledge* of such a polynomial (which SNARK applications rely upon). These are much stronger properties than originally considered in [KZG10] and in this work, typically requiring strong (often unfalsifiable) assumptions, and our construction does not achieve them. In addition, many works like [BBB$^+$18, VP19, BFS20, BDFG21, Lee21, BNO21, KPV22] allow openings to be *interactive* proofs, then heuristically (and unfalsifiably) make them noninteractive using a random oracle using Fiat–Shamir [FS86]. Our construction is natively noninteractive without any heuristics.

Scheme (Assumption)	**Setup**	**PQ**	$\lvert pp\rvert$	$\lvert c\rvert$	$\lvert \pi\rvert$
[CF13] (RSA, ECDH)	Private	✗	D, D^2	1	1
[CPSZ18] (q-SBDH)	Private	✗	D	1	1
[PSTY13] (SIS)	Public	✓	$\log^2 S$	$\log S$	$dh \log^2 S$
[PPS21] (SIS)	Private	✓	$d^2 \log S + d \log^2 S$	$\log S$	$h \log^2 S$
Our construction (SIS)	Public	✓	$\log D \log^2 S$	$\log S$	$\log D \log^2 S$

Table 1: Comparison to prior *statelessly updateable* vector commitment schemes for bit vectors of dimension $D = d^h$ (where d, h can be set freely), allowing updates to at most S entries (including the initial commitment). Here 'Setup' is either private coin (i.e., structured random string) or public coin (i.e., transparent, uniformly random string), 'PQ' indicates post-quantum security, $|pp|$ is the public parameter size, $|c|$ is the commitment size, and $|\pi|$ is the proof size. Logarithmic factors in $\log D = h \log d$ and quasi-linear factors in the security parameter are omitted throughout.

Scheme	**Assumption** (type)	**Setup**	$\lvert pp\rvert$	$\lvert c\rvert$	$\lvert \pi\rvert$	T_{Commit}	T_{Open}	T_{Verify}
[BCFL22]	twin-k-M-SIS (new)	Private	w^5	1	1	$\lvert u\rvert$	$\lvert C_u(\cdot)\rvert$	$\lvert C_u(\cdot)\rvert$
[WW23]	BASIS$_{\mathsf{struct}}$ (new)	Private	$\lvert u\rvert^2$	1	1	$\lvert u\rvert^2$	$\lvert C_u(\cdot)\rvert$	$\lvert C_u(\cdot)\rvert$
Construction 3.5	SIS (standard)	Public	$\lvert v\rvert$	1	$\lvert v\rvert$	$\lvert C_u(\cdot)\rvert$	$\lvert C_u(\cdot)\rvert$	$\lvert v\rvert$

Table 2: Comparison to concurrent works [BCFL22, WW23] that also construct functional commitments for non-linearizable functions, from falsifiable lattice assumptions. For a fair comparison, all quantities are given for the following setting: a two-input circuit $C(\cdot, \cdot)$ of width w is fixed ahead of time, where each input may represent either 'data' or 'code'. Its first input u is committed, then a proof is generated and verified for the evaluation of $C_u(\cdot) := C(u, \cdot)$ at its second input v. Here 'Setup', $|pp|$, $|c|$, and $|\pi|$ are as in Table 1, and T refers to the running time of the operation in the subscript. Dependencies on the security parameter and circuit depth, and poly-logarithmic factors in all parameters, are omitted. In addition, [BCFL22] supports a *sublinear*-time verification, following an input-independent preprocessing step.

1.2 Concurrent Related Work

We briefly discuss two concurrent works [BCFL22, WW23] that also construct functional commitments from lattice-style assumptions; see Table 2 for a summary. Both of these works require private setup of a *structured random string* (with a trapdoor) by a trusted party—versus our transparent setup—and rely on *new, ad-hoc lattice assumptions*—versus our reliance solely on the standard SIS problem, which has been well studied and is supported by worst-case-hardness theorems [Ajt96, MR04].

Regarding efficiency, their proofs are succinct, whereas ours grow with the size of the opened data, but their runtimes are "dual" to ours in a way that puts a higher burden on the verifier. Specifically, their commitment algorithms' runtimes grow merely with the *size* of the committed data, but their verifiers' runtimes grow with the *function's runtime* (though [BCFL22] also supports sublinear-time verification, after preprocessing that grows with the function's runtime). In our work, this profile is essentially reversed: the committer's runtime grows with the function's runtime (though this can be reduced in many important cases using preprocessing), but the verifier's runtime grows only quasi-linearly with the size of the opened data.

The work of [BCFL22] constructs functional commitments with private setup from a new "twin-k-MSIS" assumption, which extends an assumption recently introduced in [ACL$^+$22]. In this construction, the sizes of some objects grow with the *width* of the function's arithmetic circuit, in addition to the depth. More specifically, the public parameters scale polynomially with the width, the commitment size scales logarithmically with the width, and the proofs scale linearly in the depth.

The work of [WW23] constructs functional commitments with private setup from a new "basis-augmented" SIS-like assumption. As in our work, this construction uses homomorphic techniques from [GSW13, GVW15b], and a variant enjoys zero knowledge (which [WW23] calls "private openings"), but only with the weaker notion of *target* binding that requires the commitment to be honestly generated. In contrast to our work, the new basis-augmented SIS assumption enables succinct proofs (analogously to [PPS21]), and an extension of the public parameters enables *proof aggregation*, which allows multiple proofs (for the same commitment) to be combined into a single, shorter proof that verifies the original collection of openings.

Related notions. We also mention two other concurrent works [GSW23, KLVW23] that construct primitives that are closely related to functional commitments.

The work of [GSW23] constructs a non-interactive, publicly verifiable *delegation* scheme for committed programs, with private setup from the Learning With Errors (LWE) assumption [Reg05], via non-interactive batch arguments [CJJ21]. In this primitive, a delegator publishes a succinct digest of a program, which can later be used to verify proofs of correctness for input-output pairs, much like in functional commitments. The efficiency requirements are also essentially the same as those for functional commitments, so the two concepts are syntactically interchangeable. However, the security notion considered in [GSW23] assumes that the program digest is *honestly generated*, so it is strictly weaker than (full) evaluation binding for functional commitments, but it does match the weaker notion of target binding. In summary, functional commitments provide an alternative solution to the delegation problem considered in [GSW23].

The work of [KLVW23] constructs SNARGs for *RAM computations*, with private setup from the LWE assumption, via (somewhere-extractable) batch arguments. This primitive concisely commits to a large memory, then proves the output of a RAM program (with that memory) on a small explicit input. This is effectively a functional commitment (in "dual" form) for RAM programs, which, in contrast to the circuit model adopted in this work, can access parts of a large memory *without* having to read all of it as input.

1.3 Technical Overview

Here we summarize our general functional commitment construction and some of its important instantiations. The details are in Sections 3 and 4, respectively.

1.3.1 Functional Commitment Scheme

At a high level, our functional commitment construction works similarly to the one from [PPS21], but ours does not require an authority or even any structured public parameters. To set the stage, we briefly recall the main ideas from the construction of [PPS21] (adapted so that it commits to functions and opens at inputs).

The public parameters are a uniformly random SIS matrix $\mathbf{A}$, which is generated together with a secret "trapdoor" as in [MP12], along with another uniformly random matrix $\mathbf{C}$ whose width is proportional to the length of a function input. We view $\mathbf{C}$ as a commitment to an *as-yet undetermined* input x, under the fully homomorphic encryption/commitment scheme of [GSW13, GVW15b]. To commit to a function f, one just homomorphically evaluates f on $\mathbf{C}$, resulting in some $\mathbf{C}_f$. Accordingly, we view this as a commitment to the function value $f(x)$, but for an unspecified x.

Recall that opening a committed function at a desired input x requires an authority-generated "opening key" for x. This key is some "short" random coins that open $\mathbf{C}$ as a commitment to x, which the adversary samples using its trapdoor for $\mathbf{A}$. Given these coins, and using the properties of the homomorphic commitment scheme, the committer can track how the coins combine and grow during the homomorphic evaluation of f on $\mathbf{C}$, which results in some fairly short *derived* coins that open $\mathbf{C}_f$ as a commitment to $f(x)$. These coins serve as the proof for the function value $f(x)$. For other inputs x', the authority must generate corresponding opening keys upon request, which, again, are random coins opening $\mathbf{C}$ as a commitment to those x'.

The main challenge in implementing the above strategy is that it is actually *insecure* to equivocate $\mathbf{C}$ as a commitment to two different values—at least *relative to the same "base" matrix* $\mathbf{A}$. The solution given in [PPS21] is as follows: when the authority opens $\mathbf{C}$ as a commitment to x, it does so relative to a "tagged" base matrix $\mathbf{A}_x$ that is derived from $\mathbf{A}$ using x as the tag. This turns out to be secure to do for essentially unlimited values of x. Very importantly, the homomorphic evaluations of functions f on $\mathbf{C}$ do not depend at all on the base matrix, so they can still be computed independent of any specific x.

Our approach. As already mentioned, our functional commitment scheme works similarly, but does not use an authority or any trapdoored public parameters. Interestingly, we achieve this in a remarkably simple way, by relying on *fewer* features of the underlying homomorphic commitment scheme from [GSW13, GVW15b]. In particular, we do not use any notions of "base" matrices, commitment coins, equivocation, or even committed data at all! We refer to the stripped-down set of features that we do use as a *homomorphic computation* scheme, to reflect the fact that it does not operate on any hidden data. (See Section 3.1 for full details.)

In our scheme, the public parameter is just a uniformly random matrix $\mathbf{C}$ as above; there is no longer any trapdoored base matrix $\mathbf{A}$. In contrast to [PPS21], we do not view $\mathbf{C}$ as a commitment to any data, and never open it as such. But to commit to a function f, we still homomorphically evaluate f on $\mathbf{C}$, yielding a commitment $\mathbf{C}_f$. To open the committed function at an input x, we cannot use any opening of $\mathbf{C}$. Instead, we track an "input aware" homomorphic evaluation of f on $\mathbf{C}$, which is "shifted" by (a suitable matrix encoding of) the input x. By the properties of homomorphic computation, the result of this evaluation turns out to be $\mathbf{C}_f$, but similarly shifted by (the encoding of) $f(x)$. Moreover, this evaluation yields a fairly short multiplier matrix that links the shifted $\mathbf{C}$ and $\mathbf{C}_f$ matrices; this multiplier serves as the proof for the function value $f(x)$.

Overall, our functional commitment scheme significantly simplifies the one from [PPS21], by disposing of the trapdoored setup, the online generation of opening keys, and the tracing of commitment randomness, but it otherwise works quite similarly. One additional difference is that in our scheme, the dimension of a proof is linear in the size of the input x (because the proof is left-multiplied by the shifted $\mathbf{C}$ matrix), whereas in [PPS21] the proof dimension is essentially independent of the input size (because the proof is left-multiplied by the trapdoored matrix $\mathbf{A}_x$). This has some implications for proof size and efficiency, but proofs are still concise when the input is smaller than the function description, which is almost always the case in applications.

1.3.2 Instantiations: Key-Value Commitments, Polynomial Commitments, and More

In Section 4 we give several concrete instantiations of our general functional commitment scheme, for specific function families of interest. Most of these instantiations fit the following template: we identify a (potentially huge) set of (potentially complex) "basis" functions, and show how to express any member of the family relatively simply in terms of this basis, e.g., as a linear or low-degree combination of not too many basis functions. Using our functional commitment scheme's generic composition properties, this immediately yields a commitment scheme for the family, whose parameters and complexity are mainly determined by the homomorphic implementation of the basis functions. Moreover, if there are not too many basis functions (and similarly, function inputs that might be opened), then commitments to all of them (and their associated openings) can be preprocessed, making the "online" cost of committing (and opening) fairly low. Finally, this approach naturally supports stateless updates, e.g., by other combinations of the basis functions. We next mention some specific examples of this template that we work out in detail.

Bounded-support functions. In Section 4.1 we consider arbitrary functions of *bounded support* over some potentially huge domain $\mathcal{X}$. We show how bounded-support commitments directly yield commitments to arbitrary *key-value maps* (with a bounded number of keys), a notion introduced in [BBF19] that generalizes and unifies both vector commitments and accumulators (see also [AR20, CEO22]). A suitable basis for bounded-support functions is the set of *point functions* $\mathsf{Eq}_{\bar{x}} \colon \mathcal{X} \to \{0,1\}$ for $\bar{x} \in \mathcal{X}$, where $\mathsf{Eq}_{\bar{x}}(x)$ is defined to be 1 if $x = \bar{x}$ and 0 otherwise. Any function f of support $\mathrm{supp}(f)$ can be expressed as a linear combination of $|\mathrm{supp}(f)|$ such point functions, as $f(x) = \sum_{\bar{x} \in \mathrm{supp}(f)} f(\bar{x}) \cdot \mathsf{Eq}_{\bar{x}}(x)$.

From the above, we immediately get key-value commitments from any homomorphic implementation of the $\mathsf{Eq}_{\bar{x}}$ functions. We give a (to our knowledge) new, very simple, and low-expansion implementation, which just does $k = \log \mathcal{X}$ homomorphic bit multiplications, scheduled in a certain way. Due to the particular properties of the homomorphic computation scheme, the associated expansion factor is only *linear* in k, which ultimately leads to good SIS parameters.

Polynomials. In Section 4.2 we apply the template to obtain commitments for (univariate or multivariate) polynomials of bounded degree. Here the "basis" functions are just the powers of the input variable(s), and each such polynomial can be expressed as a linear combination of these powers (or their products, in the multivariate case). Here the linearity is over the polynomial's coefficient ring, so even if commitments to the powers are precomputed, the "online" commitment procedure does not necessarily consist solely of linear homomorphic operations. However, we show that if the coefficient ring can be embedded as a suitable matrix ring, then the online phase can be made linear, at the expense of sacrificing further multiplicative (but not linear) compositions.

1.4 Future Work

We believe that our work opens many avenues for interesting further research. One exciting direction is to see whether our techniques can pave the way for *succinct noninteractive arguments (of knowledge)* (SNARGs or SNARKs) from lattice assumptions that are simple to state and analyze. Recent work [ACL$^+$22] took a major step forward on lattice-based SNARGs, but under rather complicated and ad-hoc (knowledge) assumptions. Our functional commitments do not directly yield SNARGs because, while a commitment binds the committer to *some function*, it does not guarantee anything about the *form* of that function. In other words, nothing ties the committed function to the computation or relation for which we seek a SNARG. Indeed, SIS-based functional commitments cannot yield a complete solution, because SNARGs for NP cannot be constructed based on any falsifiable assumption, via a black-box security reduction [GW11].

Another direction for future work is to construct *subvector* commitments from standard lattice assumptions. Such commitments allow a user to commit to a vector and then later open any subset of its entries, where for nontriviality the proof size should be sublinear in the size of the subset. Our work does not achieve this because we simply prove and verify each function output independently. However, it seems likely that, due to the simple linear nature of our verifier, multiple proofs could be compressed using amortization techniques for interactive arguments for lattice relations [BBC$^+$18].

Finally, there are a number of technical improvements one could seek for our functional commitment scheme. As mentioned above, the proof size grows at least linearly with the length of the function input, and it would be good to reduce this dependence. (On the other extreme, the proof length in the functional commitment scheme of [PPS21] is essentially independent of the input length, thanks to the online authority.) Another area of potential improvement is in the notion of security achieved. We mainly prove *selective* binding, where the adversary must name the input on which it will attempt to break binding *before* seeing the public parameters. We can immediately obtain the more realistic notion of *adaptive* binding via complexity leveraging, assuming that SIS is hard to break with inverse-subexponential probability $\exp(-\lambda^{\varepsilon})$, for some (arbitrarily small) constant $\varepsilon > 0$. However, obtaining adaptive security under a tighter reduction would be welcome.

2 Preliminaries

For any non-negative integer i, denote $[i] = \{1, \dots, i\}$. For a real vector $\mathbf{v}$, let $\|\mathbf{v}\|_1 := \sum_i |v_i|$ denote its ℓ_1 norm. For a real matrix $\mathbf{V}$, let $\|\mathbf{V}\|_1 := \max_j \|\mathbf{v}_j\|$ denote the maximum $\ell_1$norm of its column vectors $\mathbf{v}_j$. Observe that for any matrices $\mathbf{V}, \mathbf{U}$, we have $\|\mathbf{VU}\|_1 \leq \|\mathbf{V}\|_1 \cdot \|\mathbf{U}\|_1$ by the triangle inequality.

The *Kronecker product* $\mathbf{A} \otimes \mathbf{B}$, where each of $\mathbf{A}, \mathbf{B}$ is a vector or a matrix, is obtained by replacing each entry $a_{i,j}$ of $\mathbf{A}$ with the block $a_{i,j}\mathbf{B}$. It obeys the *mixed-product* property: $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD})$ for any $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ having compatible dimensions.

2.1 Short Integer Solution

We briefly recall the Short Integer Solution (SIS) and its hardness based on worst-case lattice problems. Due to the particulars of our constructions and analyses, we define the problem in terms of the ℓ_1 norm, not the ℓ_2 norm (as is more typical). Because the ℓ_1 norm of any vector is at least its ℓ_2 norm, the variant of SIS that uses ℓ_1 is no easier than, and is plausibly even harder than, the one that uses ℓ_2 (for the same norm bound β).

Definition 2.1. The normal-form $\mathrm{SIS}_{n,q,m,\beta}$ problem in the ℓ_1 norm is: given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, find a non-zero integral vector $\mathbf{z} = (\mathbf{x} \in \mathbb{Z}^m, \mathbf{e} \in \mathbb{Z}^n)$ such that $\mathbf{Ax} = \mathbf{e} \pmod{q}$ and $\|\mathbf{z}\|_1 \leq \beta$.

When $q \geq \beta \cdot \tau(n)$ for a sufficiently large $\tau(n) = \tilde{O}(\sqrt{n})$ and m is polynomial in n and $\log q$, solving normal-form $\mathrm{SIS}_{n,q,m,\beta}$ (in ℓ_2, and hence also in ℓ_1) is at least as hard as approximating certain worst-case lattice problems on n-dimensional lattices to within a $\beta \cdot \tilde{O}(\sqrt{n})$ factor [Ajt96, MR04, GPV08].

2.2 Functional Commitments

Here we give a general definition of functional commitments and their main security property. In this definition, one *commits to a function* and then can *open it (with proof) at desired inputs*. This is the most natural formulation for our construction (Section 3) and most of its instantiations (see Section 4), and it naturally generalizes other notions of concise commitments, such as vector and polynomial commitments. However, if desired, the roles of the function and the input can be swapped via the standard technique of using a universal function (see Section 4.3).

Definition 2.2. A *functional commitment scheme* for a function family $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$, where $\mathcal{X}$ is a set of *opening inputs* and $\mathcal{Y}$ is the space of *outputs* (and all of $\mathcal{F}, \mathcal{X}, \mathcal{Y}$ may depend on the security parameter), is a tuple of algorithms with the following interfaces:

- $\mathsf{Setup}()$, given an (implicit) security parameter, outputs *public parameters* pp.
- $\mathsf{Commit}(pp, f \in \mathcal{F})$, given public parameters pp and a function f, outputs a *commitment* c_f to f.

- $\mathsf{Open}(pp, f, x \in \mathcal{X})$, given public parameters pp, function f, and an opening input x, outputs a *proof* $p_{f,x}$ attesting to the value of $f(x)$.
- $\mathsf{Verify}(pp, c_f, x \in \mathcal{X}, y \in \mathcal{Y}, p_{f,x})$, given public parameters pp, some c_f that purportedly commits to some function f, an opening input x, a claimed value y, and a purported proof $p_{f,x}$ that $f(x) = y$, either accepts or rejects.

The scheme should satisfy the following correctness property: for any $f \in \mathcal{F}$ and $x \in \mathcal{X}$, and for any $pp \leftarrow \mathsf{Setup}()$, $c_f \leftarrow \mathsf{Commit}(pp, f)$, and $p_{f,x} \leftarrow \mathsf{Open}(pp, f, x)$, $\mathsf{Verify}(pp, c_f, x, f(x), p_{f,x})$ should accept. For nontriviality, commitments should be *concise*, i.e., smaller than the representations of functions from the family $\mathcal{F}$.

Definition 2.3. For a functional commitment scheme FCS (or more precisely, just its Verify algorithm), the *selective-input attack* game between an adversary and a challenger is defined as follows:

1. The adversary is given the security parameter λ and outputs an opening input $x^* \in \mathcal{X}$ to the challenger.
2. The challenger lets $pp \leftarrow \mathsf{Setup}()$ and gives pp to the adversary.
3. Finally, the adversary outputs a commitment c^* and two value-proof pairs (y_0, p_0) and (y_1, p_1). It *wins* the game if $y_0 \neq y_1$, and if $\mathsf{Verify}(pp, c^*, x^*, y_b, p_b)$ accepts for both $b \in \{0, 1\}$.

The advantage of an adversary $\mathcal{A}$ in the above game, denoted $\mathbf{Adv}^{\mathsf{sia}}_{\mathsf{FCS}}(\mathcal{A})$, is the probability that it wins the game (as a function of the security parameter). We say that FCS (or just its Verify algorithm) has *(selective) evaluation binding* if $\mathbf{Adv}^{\mathsf{sia}}_{\mathsf{FCS}}(\mathcal{A}) = \mathrm{negl}(\lambda)$ for every probabilistic polynomial-time adversary $\mathcal{A}$.

As explained in Section 4, (selective) evaluation binding captures the main security property for prior special cases of functional commitments, e.g., position binding for vector commitments.

Remark 2.4 (Target security). A weaker notion of security, which is relevant to our zero-knowledge variant construction, is called *target* evaluation binding. Essentially, this says that it is infeasible to prove an incorrect input-output pair relative to an *honestly generated* commitment to a function—even given any randomness used in generating the commitment. (This is sufficient for, e.g., the program-delegation problem considered in [GSW23].) Formally, we consider the following modification of Definition 2.3:

- After Step 2, the adversary outputs a target function $f^* \in \mathcal{F}$, and the challenger generates a commitment $c^* \leftarrow \mathsf{Commit}(pp, f^*)$, which it gives (along with the random coins, if any) to the adversary.
- In Step 3, the adversary outputs a value-proof pair (y^*, p^*), and wins if $\mathsf{Verify}(pp, c^*, x^*, y^*, p^*)$ accepts but $y^* \neq f^*(x^*)$.

Remark 2.5 (Adaptive security). One can strengthen Definition 2.3 (and its target variant) by changing the attack game so that the adversary does not specify the target opening input x^* until Step 3 (rather than in Step 1, before seeing the public parameters); we call the resulting security notion *adaptive*, or *full*, evaluation binding. Generically, any scheme with selective security also has adaptive security, up to a loose reduction whose advantage is smaller by a factor of $|\mathcal{X}|$, the size of the input space. This follows by the standard technique of complexity leveraging—i.e., initially "guessing" the input x^* that the adversary will eventually choose. Concretely, we can obtain adaptive security for Construction 3.5 under the assumption that SIS is hard to break with inverse-subexponential probability $\exp(-\lambda^{\varepsilon})$, for some (arbitrarily small) constant $\varepsilon > 0$.

3 Functional Commitments from SIS

In this section we construct a very general functional commitment scheme, supporting rich and complex function classes, based on the SIS problem.

3.1 Homomorphic Computation

The heart of our functional commitment scheme is what we call a "homomorphic computation" scheme, which is inherent in the homomorphic encryption scheme of Gentry, Sahai, and Waters (GSW) [GSW13], and was made more explicit in the works of [BV14, AP14, GVW15b], with other useful properties derived in related works like [BGG$^+$14, GVW15a, PS19, PPS21].

Here we lay out a somewhat different perspective that focuses on just those limited properties needed for our purposes, and hence exposes less of the functionality than is used in prior works, making it slightly simpler. In particular, we do not need any explicit notions of encrypted/committed/hidden data, encryption/commitment randomness, or decryption/opening.

3.1.1 Overview

The homomorphic computation scheme operates on matrices $\mathbf{C} \in \mathbb{Z}_q^{n \times W}$ for some fixed q, n and various W. Performing homomorphic operations that correspond to a function f yields a matrix $\mathbf{C}_f \in \mathbb{Z}_q^{n \times W'}$. The key property is that for *any* input x to f,

$$(\mathbf{C} - \mathsf{Rep}(x) \otimes \mathbf{g}^t) \cdot \mathbf{S}_{f,x} = \mathbf{C}_f - \mathsf{Rep}(f(x)) \otimes \mathbf{g}^t \tag{3.1}$$

for some "short" (and efficiently computable) matrix $\mathbf{S}_{f,x} \in \mathbb{Z}^{W \times W'}$ that can depend on $\mathbf{C}, f, x$. Here Rep outputs a suitable matrix representation of its argument, and $\mathbf{g}$ is a special fixed vector, described next.

The above Kronecker products serve as "robust" matrix encodings using a fixed *gadget* vector $\mathbf{g} \in \mathbb{Z}_q^{\ell}$, which must come with a corresponding *decomposition* function $\mathbf{g}^{-1} \colon \mathbb{Z}_q \to \mathbb{Z}^{\ell}$. These are defined so that $\mathbf{g}^{-1}(u) \in \mathbb{Z}^{\ell}$ is "short" (relative to q) and $\mathbf{g}^t \cdot \mathbf{g}^{-1}(u) = u$, for all $u \in \mathbb{Z}_q$. This naturally extends to $(\mathbf{X} \otimes \mathbf{g}^t) \cdot \mathbf{g}^{-1}(\mathbf{Y}) = \mathbf{XY}$ for any $\mathbf{X} \in \mathbb{Z}_q^{n \times d}$ and $\mathbf{Y} \in \mathbb{Z}_q^{d \times d'}$, where $\mathbf{g}^{-1}(\mathbf{Y})$ operates entry-wise, replacing each $y_{i,j}$ with the "short" column vector $\mathbf{g}^{-1}(y_{i,j}) \in \mathbb{Z}^{\ell}$.

For concreteness, we use the powers-of-two gadget $\mathbf{g} := (1, 2, 4, \ldots, 2^{\ell-1})^t$ for $\ell = \lceil \log_2 q \rceil$, where $\mathbf{g}^{-1} \colon \mathbb{Z}_q \to \{0,1\}^{\ell}$ simply outputs the binary representation of (the distinguished representative in $\{0, 1, \ldots, q-1\}$ of) its argument, least-significant bit first. All of what follows straightforwardly generalizes to other choices of gadget, with suitable adjustments to the bounds on "short" objects.

In addition, the decomposition operation $\mathbf{g}^{-1}$ can be *randomized* as in [MP12, AP14]. This can be used for (statistically) *function-hiding* homomorphic computation, using ideas from [BPMW16], but simplified in our setting because only SIS (not LWE) instances need to be re-randomized. Function hiding is used only in the zero-knowledge variant of our functional commitment construction (see Section 3.2.6).

3.1.2 Linear Homomorphisms

We recall the homomorphic operations supporting linear functions. First, for any $\mathbf{X}_1, \mathbf{X}_2 \in \mathbb{Z}_q^{n \times d}$ (for any d) we have

$$\big([\mathbf{X}_1 \mid \mathbf{X}_2] \otimes \mathbf{g}^t\big) \cdot \underbrace{\begin{bmatrix} \mathbf{I}_{d\ell} \\ \mathbf{I}_{d\ell} \end{bmatrix}}_{\mathbf{S}_+} = \left([\mathbf{X}_1 \mid \mathbf{X}_2] \cdot \begin{bmatrix} \mathbf{I}_d \\ \mathbf{I}_d \end{bmatrix}\right) \otimes (\mathbf{g}^t \cdot \mathbf{I}_\ell) = (\mathbf{X}_1 + \mathbf{X}_2) \otimes \mathbf{g}^t.$$

This yields the homomorphic operation for addition: given any $\mathbf{C} \in \mathbb{Z}_q^{n \times 2d\ell}$, define $\mathbf{C}_+ := \mathbf{C} \cdot \mathbf{S}_+ \in \mathbb{Z}_q^{n \times d\ell}$. Then for any $\mathbf{X}_1, \mathbf{X}_2 \in \mathbb{Z}_q^{n \times d}$ we have

$$(\mathbf{C} - [\mathbf{X}_1 \mid \mathbf{X}_2] \otimes \mathbf{g}^t) \cdot \mathbf{S}_+ = \mathbf{C}_+ - (\mathbf{X}_1 + \mathbf{X}_2) \otimes \mathbf{g}^t. \tag{3.2}$$

Note that $\mathbf{S}_+$ is short: $\|\mathbf{S}_+\|_1 = 2$.

Second, recall from above that for any $\mathbf{X} \in \mathbb{Z}_q^{n \times d}$ and $\mathbf{Y} \in \mathbb{Z}_q^{d \times d'}$,

$$(\mathbf{X} \otimes \mathbf{g}^t) \cdot \underbrace{\mathbf{g}^{-1}(\mathbf{Y})}_{\mathbf{S}_{\times \mathbf{Y}}} = \mathbf{X}\mathbf{Y} \in \mathbb{Z}_q^{n \times d'}.$$

This yields homomorphic (right-)multiplication by any fixed matrix $\mathbf{Y}$: given any $\mathbf{C} \in \mathbb{Z}_q^{n \times d\ell}$, define $\mathbf{C}_{\times \mathbf{Y}} := \mathbf{C} \cdot \mathbf{S}_{\times \mathbf{Y}}$. Then for any $\mathbf{X} \in \mathbb{Z}_q^{n \times d}$, we have

$$(\mathbf{C} - \mathbf{X} \otimes \mathbf{g}^t) \cdot \mathbf{S}_{\times \mathbf{Y}} = \mathbf{C}_{\times \mathbf{Y}} - (\mathbf{X}\mathbf{Y}) \otimes \mathbf{g}^t. \tag{3.3}$$

Note that $\mathbf{S}_{\times \mathbf{Y}}$ is short: $\|\mathbf{S}_{\times \mathbf{Y}}\|_1 \le d\ell$.

Linear functions over finite fields. For values in the matrix ring $\mathcal{R} = \mathbb{Z}_q^{n \times n}$, the above yields a linearly homomorphic scheme, i.e., one that supports addition and (right-)multiplication by known $\mathcal{R}$-elements. This in turn yields a linearly homomorphic scheme for the finite field $\mathbb{F}_{p^{n'}}$ for any prime p that divides q and any $n' \le n$, using standard encoding and padding techniques. (In summary: $\mathbb{F}_{p^{n'}}$ is an n'-dimensional vector space over $\mathbb{F}_p$, so multiplication by each field element can be represented by a corresponding matrix in $\mathbb{Z}_p^{n' \times n'}$, which can be scaled and padded to $\mathbb{Z}_q^{n \times n}$.)

3.1.3 Multiplicative Homomorphism

To support multiplicative homomorphism, and thereby arbitrary Boolean circuits of bounded size or branching programs, we restrict all the data values to be bits, and represent any $b \in \{0, 1\}$ by the scaled identity matrix $b\mathbf{I}_n \in \mathbb{Z}_q^{n \times n}$. This leads to the homomorphic operation for multiplication: given any $\mathbf{C} = [\mathbf{C}_1 \mid \mathbf{C}_2]$ where each $\mathbf{C}_i \in \mathbb{Z}_q^{n \times w}$ for $w = n\ell$, define $\mathbf{C}_\times := \mathbf{C}_1 \cdot \mathbf{g}^{-1}(\mathbf{C}_2) \in \mathbb{Z}_q^{n \times w}$. Then for any $x_1, x_2 \in \{0, 1\}$, we have

$$\begin{aligned}(\mathbf{C} - [x_1\mathbf{I}_n \mid x_2\mathbf{I}_n] \otimes \mathbf{g}^t) \cdot \underbrace{\begin{bmatrix} \mathbf{g}^{-1}(\mathbf{C}_2) \\ x_1\mathbf{I}_w \end{bmatrix}}_{\mathbf{S}_{\times, x_1}} &= \mathbf{C}_\times + \mathbf{C}_2 \cdot x_1\mathbf{I}_w - x_1\mathbf{I}_n \cdot \mathbf{C}_2 - (x_2\mathbf{I}_n \otimes \mathbf{g}^t) \cdot (x_1\mathbf{I}_n \otimes \mathbf{I}_\ell) \\ &= \mathbf{C}_\times - (x_1x_2)\mathbf{I}_n \otimes \mathbf{g}^t. \end{aligned} \tag{3.4}$$

Note that the multiplier matrix $\mathbf{S}_{\times, x_1}$ is short: $\|\mathbf{S}_{\times, x_1}\|_1 \le w + 1$. Also note that, unlike above, here the multiplier matrix depends on the initial matrix $\mathbf{C}_2$ as well as one of the input values x_1 (which is not determined at the time of the homomorphic multiplication).

More generally, the asymmetric form of the short multiplier matrix $\mathbf{S}_{\times, x_1}$ means we can perform many sequential multiplications with a short multiplier whose norm bound is only *linear* in the number of operations. Specifically, given any $\mathbf{C} = [\mathbf{C}_1 \mid \cdots \mid \mathbf{C}_k]$ where each $\mathbf{C}_i \in \mathbb{Z}_q^{n \times w}$, define $\mathbf{C}_\times = \mathbf{C}_1 \cdot \mathbf{g}^{-1}(\mathbf{C}_2 \cdot \mathbf{g}^{-1}(\cdots \mathbf{g}^{-1}(\mathbf{C}_k)))$. Then for any $\mathbf{x} \in \{0, 1\}^k$, by iteratively applying the above we get a multiplier matrix

$$\mathbf{S}_{\times,\mathbf{x}} = \begin{bmatrix} \mathbf{g}^{-1}(\mathbf{C}_2 \cdot \mathbf{g}^{-1}(\mathbf{C}_3 \cdots \mathbf{g}^{-1}(\mathbf{C}_k))) \\ x_1 \cdot \mathbf{g}^{-1}(\mathbf{C}_3 \cdots \mathbf{g}^{-1}(\mathbf{C}_k)) \\ \vdots \\ x_1 \cdots x_{k-2} \cdot \mathbf{g}^{-1}(\mathbf{C}_k) \\ x_1 \cdots x_{k-1} \cdot \mathbf{I}_w \end{bmatrix} \in \mathbb{Z}^{kw \times w}. \tag{3.5}$$

Note that $\|\mathbf{S}_{\times,\mathbf{x}}\|_1 \leq (k-1)w + 1$.

Using the above, we can homomorphically evaluate any Boolean circuit f on a given matrix $\mathbf{C}$, by expressing each gate of the circuit algebraically (e.g., x NAND $y = 1 - xy$). Due to Equations (3.2) to (3.4), the result $\mathbf{C}_f$ satisfies Equation (3.1) for any circuit input x, where the multiplier matrix $\mathbf{S}_{f,x}$ is the product of some short multiplier matrices, and hence is somewhat short itself. Similarly, we can homomorphically evaluate *branching programs*, where, as with multiplication of several bits, the asymmetric nature of each step's multiplier matrix means that their product has ℓ_1 norm proportional to the program length.

3.1.4 Summary

We summarize all of the above in the following.

Definition 3.1. Define the following function families $\mathcal{F}_{\text{linear}}, \mathcal{F}_{\text{circuit}}, \mathcal{F}_{\text{BP}}$:

- $\mathcal{F}^k_{\text{linear}} = \{f_{\mathbf{w}} \colon \mathbb{F}^k \to \mathbb{F} : \mathbf{w} \in \mathbb{F}^k\}$ is the family of linear functions $f_{\mathbf{w}}(\mathbf{x}) := \langle \mathbf{w}, \mathbf{x} \rangle$ over $\mathbb{F}$, where $\mathbb{F} = \mathbb{F}_{p^{n'}}$ for any prime p that divides q and any $n' \leq n$.
- $\mathcal{F}^k_{\text{circuit}} = \{f \colon \{0,1\}^k \to \{0,1\}\}$ is the family of functions that are computable by Boolean circuits of some specified depth D (and fan-in two).
- $\mathcal{F}^k_{\text{BP}} = \{f \colon \{0,1\}^k \to \{0,1\}\}$ is the family of functions that are computable by branching programs of some specified size S (and some fixed width).

Remark 3.2. In the above-defined families, for simplicity we restrict the functions to output a single value (i.e., a field element or bit). This is without loss of generality, because any vector-valued (i.e., multi-output) function of the same complexity can be obtained as the concatenation of the functions that produce each entry of its output vector, and we can commit to and open each such function in parallel. Indeed, our concrete instantiations will use vector-valued functions, which are implicitly handled in this way.

Theorem 3.3 (Homomorphic computation scheme). *Let $n, q \in \mathbb{N}$ and $D = \{0,1\}$ or $D = \mathbb{F}_{p^{n'}}$ for a prime p that divides q and some $n' \leq n$. There is an efficient deterministic robust matrix encoding for any $\vec{v} \in D^d$, denoted $\vec{v}^t \otimes \mathbf{g}^t \in \mathbb{Z}_q^{n \times dw}$ where $w = n\ell$, and a deterministic polynomial-time homomorphic evaluation algorithm* Eval, *where for any function family $\mathcal{F} = \{f \colon D^k \to D\}$ from Definition 3.1:*[5]

- Eval*'s input in square brackets is optional, and when it is provided, the additional output (also in square brackets) is also produced. The non-optional output is unaffected by whether or not an optional input is provided.*
- Eval$(f \in \mathcal{F}, \mathbf{C} \in \mathbb{Z}_q^{n \times kw}[, \vec{x} \in D^k])$ *outputs a matrix $\mathbf{C}_f \in \mathbb{Z}_q^{n \times w}$ [and an integral matrix $\mathbf{S}_{f,\vec{x}} \in \mathbb{Z}^{kw \times w}$], where the additional output $\mathbf{S}_{f,\vec{x}}$ satisfies*

$$(\mathbf{C} - \vec{x}^t \otimes \mathbf{g}^t) \cdot \mathbf{S}_{f,\vec{x}} = \mathbf{C}_f - f(\vec{x})^t \otimes \mathbf{g}^t, \tag{3.6}$$

[5] More generally, the scheme works for vector-valued (multi-output) functions, following Remark 3.2.

and

1. *for* $\mathcal{F} = \mathcal{F}^k_{\textit{linear}}$, $\|\mathbf{S}_{f,\vec{x}}\|_1 \leq kw$; *moreover,* $\|\mathbf{S}_{f,\vec{x}}\|_1 \leq k$ *when* f *is a subset-sum function;*
2. *for* $\mathcal{F} = \mathcal{F}^k_{\textit{circuit}}$, $\|\mathbf{S}_{f,\vec{x}}\|_1 \leq O(w)^D$;
3. *for* $\mathcal{F} = \mathcal{F}^k_{\textit{BP}}$, $\|\mathbf{S}_{f,\vec{x}}\|_1 \leq w^{O(1)} \cdot S$.

Remark 3.4. The form of Equation (3.6) means that Eval is *composable*, i.e., it can be applied to its own outputs, to homomorphically compute on the results of other homomorphic computations. More specifically, we can compute $(\mathbf{C}_f \, [, \mathbf{S}_{f,\vec{x}}]) = \mathsf{Eval}(f, \mathbf{C} \, [, \vec{x}])$ then $(\mathbf{C}_{g\circ f} \, [, \mathbf{S}_{g,f(\vec{x})}]) = \mathsf{Eval}(g, \mathbf{C}_f \, [, f(\vec{x})])$, where recall that f, g can be vector-valued functions. Then $\mathbf{S}_{f,\vec{x}}, \mathbf{S}_{g,f(\vec{x})}$ satisfy the norm bounds corresponding to f, g (respectively), and

$$\begin{aligned}(\mathbf{C} - \vec{x}^t \otimes \mathbf{g}^t) \cdot \mathbf{S}_{f,\vec{x}} \cdot \mathbf{S}_{g,f(\vec{x})} &= (\mathbf{C}_f - f(\vec{x})^t \otimes \mathbf{g}^t) \cdot \mathbf{S}_{g,f(\vec{x})} \\ &= \mathbf{C}_{g\circ f} - g(f(\vec{x}))^t \otimes \mathbf{g}^t.\end{aligned}$$

For notational convenience, we express the above process (hiding the intermediate values) as

$$(\mathbf{C}_{g\circ f} \, [, \mathbf{S}_{g\circ f,\vec{x}}]) = \mathsf{Eval}(g \circ f, \mathbf{C} \, [, \vec{x}]),$$

where $\mathbf{S}_{g\circ f,\vec{x}} = \mathbf{S}_{f,\vec{x}} \cdot \mathbf{S}_{g,f(\vec{x})}$.

3.2 Functional Commitment Construction

Our functional commitment scheme is parameterized by a function family $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ and a corresponding norm bound κ, which is used only in verification. Recall from Definition 2.2 that a user first commits to some function $f \in \mathcal{F}$. Then, for one or more inputs $x \in \mathcal{X}$, the user can generate a proof that $f(x) = y$ for some claimed $y \in \mathcal{Y}$. The main security property of interest (see Definition 2.3) is essentially that it should be infeasible to generate a (possibly malformed) commitment, an input x, and valid proofs for two different purported function outputs (at x).

Construction 3.5 (SIS-based functional commitment). Let $n, q \in \mathbb{N}$. Following Theorem 3.3, let $\mathcal{X}, \mathcal{Y}$ be finite domains where $x \in \mathcal{X}, y \in \mathcal{Y}$ have robust matrix encodings denoted $x^t \otimes \mathbf{g}^t \in \mathbb{Z}_q^{n\times W}$, $y^t \otimes \mathbf{g}^t \in \mathbb{Z}_q^{n\times W'}$ (respectively), and let Eval be the homomorphic evaluation algorithm. Let $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ be some function family and κ be a corresponding norm bound.[6] Define the following functional commitment scheme for $\mathcal{F}$.

- $\mathsf{Setup}()$: choose uniformly random $\mathbf{C} \leftarrow \mathbb{Z}_q^{n\times W}$ and output it as the public parameter. (Note that this is an "unstructured" random string, so the setup is untrusted.)
- $\mathsf{Commit}(\mathbf{C}, f \in \mathcal{F})$: output commitment $\mathbf{C}_f = \mathsf{Eval}(f, \mathbf{C}) \in \mathbb{Z}_q^{n\times W'}$.[7]

  [For Boolean functions the commitment can be compressed significantly; see Section 3.2.3 below.]

[6] The norm bound κ should be set large enough so that the verifier accepts properly generated proofs for all functions in $\mathcal{F}$ (see Lemma 3.6). Then, n and q should be set so that the SIS problem underlying the scheme's evaluation binding is sufficiently hard (see Theorem 3.7).

[7] We can make the commitment hide the function f by using *circuit-private* homomorphic computation, such as from [BPMW16], and retaining the randomness for use in opening.

- $\mathsf{Open}(\mathbf{C}, f \in \mathcal{F}, x \in \mathcal{X})$: compute $(\mathbf{C}_f, \mathbf{S}_{f,x}) = \mathsf{Eval}(f, \mathbf{C}, x)$, and output proof $\mathbf{S}_{f,x} \in \mathbb{Z}^{W \times W'}$.[8] [For Boolean functions the proof can be compressed significantly; see Section 3.2.3 below.]
- $\mathsf{Verify}_\kappa(\mathbf{C}, \mathbf{C}^*, x \in \mathcal{X}, y \in \mathcal{Y}, \mathbf{S}^*)$: if $\|\mathbf{S}^*\|_1 \leq \kappa$ and
$$(\mathbf{C} - x^t \otimes \mathbf{g}^t) \cdot \mathbf{S}^* = \mathbf{C}^* - y^t \otimes \mathbf{g}^t,$$
then accept; otherwise, reject.

Section 3.2.6 describes a zero-knowledge variant of this construction with the weaker notion of *target* evaluation binding, and sketches the proofs of its security properties.

3.2.1 Complexity

For security (see Theorem 3.7 and the remark following Definition 2.1), the modulus q should satisfy $q \geq \kappa \cdot \tau(n)$, or $q \geq k'\kappa \cdot \tau(n)$ for the compressed variant for k'-bit function outputs, for some large enough $\tau(n)$ that can be $\tilde{O}(\sqrt{n})$. Because $\kappa \geq \sqrt{n}$ in any useful instantiation, we can ensure that $\log_2 q = \Theta(\log \kappa)$. Recall that $w = n\ell = n\lceil \log_2 q \rceil$.

The sizes of the scheme's various objects are as follows:

- The public parameter $\mathbf{C}$ is $\approx W \cdot w$ bits, where W is the width of the robust matrix encoding of any input $x \in \mathcal{X}$. Concretely, $W = kw$ when $\mathcal{X} = D^k$ for $D = \{0, 1\}$ or $D = \mathbb{F}_{p^{n'}}$ (see Definition 3.1), so the public parameter is $\approx kw^2$ bits.
- An (uncompressed) commitment $\mathbf{C}_f$ is $\approx W' \cdot w$ bits, where W' is the width of the robust matrix encoding of any output $y \in \mathcal{Y}$, which is $W' = k'w$ when $\mathcal{Y} = D^{k'}$. So, a commitment is $\approx k'w^2$ bits. A compressed commitment for k'-bit function outputs (where $k' \leq n$) is just $\approx w$ bits; see Section 3.2.3 below.
- An (uncompressed) proof $\mathbf{S}_{f,x}$ is $\approx W \cdot W' \cdot \log_2 \kappa$ bits (using a naïve encoding); for a $D^k \to D^{k'}$ function, this is $\approx k \cdot k' \cdot w^2 \cdot \log_2 \kappa$ bits. A compressed proof for k'-bit function outputs (where $k' \leq n$) is just $kw \log_2(k'\kappa)$ bits.

(We can reduce the sizes of all these objects by about a factor of n, using "algebraically structured" lattices and the Ring-SIS problem [Mic02, PR06, LM06].)

The running times of Commit and Open are just those of homomorphic evaluation of the committed function, in the latter case with the known input. The verifier's running time is dominated by a matrix multiplication, or a matrix-vector multiplication for a compressed proof.

3.2.2 Composition, Stateless Updates, and (Outsourced) Precomputation

Because both algorithms $\mathsf{Commit}, \mathsf{Open}$ are simply the homomorphic evaluation algorithm Eval (with the latter providing the function input x as Eval's optional input), Construction 3.5 supports function composition in the same way that the homomorphic computation scheme does, as described in Remark 3.4. We use similar notation $\mathbf{C}_{g \circ f} = \mathsf{Commit}(\mathbf{C}, g \circ f)$ and $\mathbf{S}_{g \circ f, x} = \mathsf{Open}(\mathbf{C}, g \circ f, x)$ to denote this kind of composition. Note that f and g need not come from the same function family. As usual, for correctness we simply need to use an appropriate norm bound κ in verification.

[8] In concert with a function-hiding commitment, we can make the opening reveal nothing more than the single input-output pair $(x, f(x))$ by giving a zero-knowledge proof (of knowledge) of some $\mathbf{S}_{f,x}$ that satisfies the verifier.

This kind of composition has several beneficial consequences: it enables *stateless updates*, *reuse*, and *(outsourced) precomputation* of commitments and proofs. More specifically, a commitment $\mathbf{C}_f$ to a function f can be updated to a commitment $\mathbf{C}_{g\circ f} = \mathsf{Commit}(\mathbf{C}_f, g)$ to any $g \circ f$, using $\mathbf{C}_f$ and g alone; the original function f is not needed. Similarly, for any x, a proof $\mathbf{S}_{f,x}$ that $f(x) = y$ can be updated to a proof $\mathbf{S}_{g\circ f,x} = \mathbf{S}_{f,x} \cdot \mathsf{Open}(\mathbf{C}_f, g, y)$ that $g(f(x)) = z$, using just $\mathbf{C}_f, g, y$. In addition, commitments and proofs for other functions $g' \circ f$ can be created by reusing $\mathbf{C}_f$ and its proofs $\mathbf{S}_{f,x}$. This enables precomputation, in case a committer does yet not know which of multiple functions with common structure $g_i \circ f$ it will commit to, or which inputs it will open. Finally, this precomputation of commitments and proofs can even be outsourced to an untrusted worker, as long as the client verifies them. Then any commitments and proofs derived (exclusively) from them will verify as well.

Several of our concrete instantiations in Section 4 rely on a few specific kinds of compositions. Fix some "base" function f. Then for an "additive update" function δ, the updated function $f' = f + \delta$ obviously satisfies $f'(\bar{x}) = f(\bar{x}) + \delta(\bar{x})$ for all $\bar{x}$. For a "multiplicative update" function c (which can be, but need not be, a constant), the updated function $f' = c \cdot f$ satisfies $f'(\bar{x}) = c(\bar{x}) \cdot f(\bar{x})$ for all $\bar{x}$. Most generally, for a "post-processing update" function g, the updated function $f' = g \circ f$ satisfies $f'(\bar{x}) = g(f(\bar{x}))$ for all $\bar{x}$.

3.2.3 Compression for Binary Functions

For concatenations of any $k' \leq n$ functions with Boolean (or more generally, small integer) outputs, we can significantly reduce the sizes of the commitments and proofs, by a factor of $W' = k'w = k'n\ell$.

Let $\mathbf{e} = (\mathbf{e}_1^t, \mathbf{e}_2^t, \dots, \mathbf{e}_{k'}^t)^t \otimes \mathbf{g}^{-1}(1) \in \{0,1\}^{k'w}$, where $\mathbf{e}_i \in \mathbb{Z}_q^n$ is the ith standard basis vector, and note that $\|\mathbf{e}\|_1 = k'$. Then any commitment $\mathbf{C}_f \in \mathbb{Z}_q^{n\times W'}$ can be compressed as the single vector $\mathbf{c}_f = \mathbf{C}_f \cdot \mathbf{e} \in \mathbb{Z}_q^n$, and any proof $\mathbf{S}_{f,x} \in \mathbb{Z}^{W\times W'}$ can be replaced by a single vector $\mathbf{s}_{f,x} = \mathbf{S}_{f,x} \cdot \mathbf{e} \in \mathbb{Z}^W$.[9] We then define the compressed verification algorithm $\mathsf{Verify}'_\kappa(\mathbf{C}, \mathbf{c}^*, x, \mathbf{y} \in \{0,1\}^{k'}, \mathbf{s}^*)$ to accept if $\|\mathbf{s}^*\|_1 \leq \kappa' := k'\kappa$ and

$$(\mathbf{C} - x^t \otimes \mathbf{g}^t) \cdot \mathbf{s}^* = \mathbf{c}^* - \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix}.$$

In brief, this compressed scheme is correct because the uncompressed scheme is, and because $\|\mathbf{s}_{f,x}\|_1 \leq \|\mathbf{S}_{f,x}\|_1 \cdot \|\mathbf{e}\|_1 \leq k'\kappa = \kappa'$ and $(\mathbf{y}^t \otimes \mathbf{I}_n \otimes \mathbf{g}^t) \cdot \mathbf{e} = \left(\begin{smallmatrix} \mathbf{y} \\ \mathbf{0} \end{smallmatrix}\right)$ by construction of $\mathbf{e}$. See Lemma 3.6 below for details, and for security see Theorem 3.7.

Compressed commitments and proofs are no longer generally composable, i.e., they do not support *arbitrary* further homomorphic operations. However, it is straightforward to see that they still are linear homomorphic via small integer combinations, as long as the norm bound used in verification is set appropriately.

3.2.4 Correctness

Lemma 3.6. *For the values of κ given above, Construction 3.5 [and its compressed variant for Boolean functions] is a correct functional commitment scheme for the corresponding function family.*

Proof. Let $f \in \mathcal{F}$ and $x \in \mathcal{X}$ be arbitrary, and let $\mathbf{C} \leftarrow \mathsf{Setup}()$ and $(\mathbf{C}_f, \mathbf{S}_{f,x}) = \mathsf{Open}(\mathbf{C}, f, x) = \mathsf{Eval}(f, \mathbf{C}, x)$. Note that $\mathbf{C}_f = \mathsf{Commit}(\mathbf{C}, f) = \mathsf{Eval}(f, \mathbf{C})$ by definition of Eval.

We show that $\mathsf{Verify}_\kappa(\mathbf{C}, \mathbf{C}_f, x, f(x), \mathbf{S}_{f,x})$ accepts. By the correctness of Eval (Equation (3.6)), we have

$$(\mathbf{C} - x^t \otimes \mathbf{g}^t) \cdot \mathbf{S}_{f,x} = \mathbf{C}_f - f(x)^t \otimes \mathbf{g}^t.$$

[9] In some contexts, compressed commitments and proofs even be computed directly, without first computing uncompressed ones.

In addition, $\|\mathbf{S}_{f,x}\|_1 \leq \kappa$ by Theorem 3.3, so Verify_κ accepts.

For the compressed variant for k'-bit outputs, where $W' = k'w$, the commitment is $\mathbf{c}_f := \mathbf{C}_f \cdot \mathbf{e}$ and proof is $\mathbf{s}_{f,x} := \mathbf{S}_{f,x} \cdot \mathbf{e}$, where $\mathbf{e} \in \{0,1\}^{W'}$ is as defined in Section 3.2.3. Consider $\mathsf{Verify}'_\kappa(\mathbf{C}, \mathbf{c}_f, x, \mathbf{y} = f(x) \in \{0,1\}^{k'}, \mathbf{s}_{f,x})$. Because $f(x)^t \otimes \mathbf{g}^t = \mathbf{y}^t \otimes \mathbf{I}_n \otimes \mathbf{g}^t$ in this setting, we have that

$$(\mathbf{C} - x^t \otimes \mathbf{g}^t) \cdot \mathbf{s}_{f,x} = (\mathbf{C} - x^t \otimes \mathbf{g}^t) \cdot \mathbf{S}_{f,x} \cdot \mathbf{e} = \big(\mathbf{C}_f - (\mathbf{y}^t \otimes \mathbf{I}_n \otimes \mathbf{g}^t)\big) \cdot \mathbf{e} = \mathbf{c}_f - \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix}$$

and $\|\mathbf{s}_{f,x}\|_1 \leq \|\mathbf{S}_{f,x}\|_1 \cdot \|\mathbf{e}\|_1 \leq k'\kappa = \kappa'$, so Verify'_κ accepts. □

3.2.5 Security

Theorem 3.7. *For any $\kappa > 0$, the Verify_κ algorithm from Construction 3.5 [and its compressed variant Verify'_κ for k'-bit function outputs, from Section 3.2.3] has selective evaluation binding (Definition 2.3) if normal-form $SIS_{n,q,W,\beta}$ in the ℓ_1 norm is hard, where $\beta = 2w\kappa + n$ for finite-field function outputs, and $\beta = 2\kappa + n$ for Boolean function outputs [or for the compressed variant, $\beta = 2k'\kappa + n$].*[10]

More specifically, for any adversary $\mathcal{A}$ against the selective evaluation binding of the scheme, there is a normal-form $SIS_{n,q,W,\beta}$ adversary $\mathcal{B}$ for which

$$\mathbf{Adv}^{SIS}(\mathcal{B}) \geq \mathbf{Adv}^{sia}(\mathcal{A}),$$

and whose running time is that of $\mathcal{A}$ plus a small polynomial in n.

Proof. Let $\mathcal{A}$ be any adversary that attacks the selective evaluation binding (Definition 2.3) of Verify_κ [or Verify'_κ]. For the former (non-compressed) version, we assume that the function output is a single finite-field element or bit, and hence $W' = w$, because breaking binding for multi-output functions requires breaking it at some single position.

We construct a reduction $\mathcal{B}$ which, on input $\mathbf{A} \in \mathbb{Z}_q^{n \times W}$, attempts to output a vector $\mathbf{x} \in \mathbb{Z}^W$ such that $\mathbf{A}\mathbf{x} \in \{0, \pm 1\}^n \setminus \{\mathbf{0}\} \subseteq \mathbb{Z}_q^n$ where $\|\mathbf{x}\|_1 \leq \beta - n$; note that such an $\mathbf{x}$ is a normal-form SIS solution (in ℓ_1) for $\mathbf{A}$. It operates as follows:

1. Give the security parameter to $\mathcal{A}$ and receive $x^* \in \mathcal{X}$ in return.
2. Let the public parameter $\mathbf{C} := \mathbf{A} + (x^*)^t \otimes \mathbf{g}^t$ and give it to $\mathcal{A}$. Note that this makes $\mathbf{C} - (x^*)^t \otimes \mathbf{g}^t = \mathbf{A}$.
3. $\mathcal{A}$ outputs some $(\mathbf{C}^*, y_0, \mathbf{S}_0, y_1, \mathbf{S}_1)$. If $y_0 = y_1$, abort.

   [For the compressed variant, $\mathcal{A}$ instead outputs some $(\mathbf{c}^*, \mathbf{y}_0, \mathbf{s}_0, \mathbf{y}_1, \mathbf{s}_1)$.]
4. Compute the binary vector $\mathbf{e} = \mathbf{g}^{-1}(\mathbf{Y}^{-1}\mathbf{e}_1) \in \{0,1\}^w$, where $\mathbf{Y} \in \mathbb{Z}_q^{n \times n}$ is the invertible matrix representing multiplication by $y_1 - y_0 \neq 0$ (and $\mathbf{e}_1 \in \mathbb{Z}_q^n$ is the first standard basis vector).

   More specifically, for finite-field outputs, $y_1 - y_0$ is a nonzero field element, hence it corresponds to an invertible $\mathbf{Y} \in \mathbb{Z}_q^{n \times n}$, and $\|\mathbf{e}\|_1 \leq w$. For Boolean functions, $y_0 - y_1 = \pm 1$, and hence corresponds to the invertible matrix $\pm \mathbf{I}_n \in \mathbb{Z}_q^{n \times n}$, so we can use $\mathbf{e} = \pm \mathbf{e}_1$ and hence $\|\mathbf{e}\|_1 = 1$.

   [For the compressed variant, this step is skipped.]
5. Output $\mathbf{x} = (\mathbf{S}_0 - \mathbf{S}_1)\mathbf{e} \in \mathbb{Z}^W$.

   [For the compressed variant, output $\mathbf{x} = \mathbf{s}_0 - \mathbf{s}_1 \in \mathbb{Z}^W$.]

[10] As mentioned in Section 3.2.3, security for the compressed variant extends to concatenations of k' functions with small integer outputs, where the additive n term in β is replaced by the maximum ℓ_1 norm of the difference between two such output vectors.

By inspection, it is clear that $\mathcal{B}$ runs in the same time as $\mathcal{A}$, plus a small polynomial. In addition, for any choice of $x^* \in \mathcal{X}$ by $\mathcal{A}$, the public parameter $\mathbf{C}$ is uniformly random (because $\mathbf{A}$ is), as needed.

We now show that if $\mathcal{A}$ successfully breaks selective evaluation binding, then $\mathcal{B}$ outputs an $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} = \mathbf{e}_1 \in \mathbb{Z}_q^n$ and $\|\mathbf{x}\|_1 \leq 2k'\kappa$. In this case, we have $y_0 \neq y_1$; $\|\mathbf{S}_0\|_1, \|\mathbf{S}_1\|_1 \leq \kappa$; and

$$\mathbf{C}^* = \mathbf{A}\mathbf{S}_0 + y_0^t \otimes \mathbf{g}^t = \mathbf{A}\mathbf{S}_1 + y_1^t \otimes \mathbf{g}^t,$$

so $\mathbf{A}(\mathbf{S}_0 - \mathbf{S}_1) = (y_1 - y_0)^t \otimes \mathbf{g}^t$. Therefore,

$$\begin{aligned}\mathbf{A}\mathbf{x} &= \mathbf{A}(\mathbf{S}_0 - \mathbf{S}_1) \cdot \mathbf{e} \\ &= ((y_1 - y_0)^t \otimes \mathbf{g}^t) \cdot \mathbf{e} = \mathbf{e}_1.\end{aligned}$$

Moreover, $\|\mathbf{x}\|_1 = \|(\mathbf{S}_0 - \mathbf{S}_1)_1 \cdot \mathbf{e}\| \leq \|\mathbf{S}_0 - \mathbf{S}_1\|_1 \cdot \|\mathbf{e}\|_1$, which by the triangle inequality is at most $2w\kappa$ for finite-field outputs, and at most 2κ for Boolean outputs, as needed.

For the compressed variant (for functions with binary outputs of length $k' \leq n$), $\mathcal{A}$ instead outputs some $(\mathbf{c}^*, \mathbf{y}_0 \in \{0,1\}^{k'}, \mathbf{s}_0 \in \mathbb{Z}^W, \mathbf{y}_1 \in \{0,1\}^{k'}, \mathbf{s}_1 \in \mathbb{Z}^W)$, and succeeds if $\mathbf{y}_0 \neq \mathbf{y}_1$; $\|\mathbf{s}_0\|_1, \|\mathbf{s}_1\|_1 \leq k'\kappa$; and

$$\mathbf{c}^* = \mathbf{A}\mathbf{s}_0 + \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{0} \end{pmatrix} = \mathbf{A}\mathbf{s}_1 + \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{0} \end{pmatrix}.$$

Since $\mathbf{x} = \mathbf{s}_0 - \mathbf{s}_1$, it immediately follows that

$$\mathbf{A}\mathbf{x} = \mathbf{A}(\mathbf{s}_0 - \mathbf{s}_1) = \begin{pmatrix} \mathbf{y}_1 - \mathbf{y}_0 \\ \mathbf{0} \end{pmatrix} \in \{0, \pm 1\}^n \setminus \{\mathbf{0}\},$$

and $\|\mathbf{x}\|_1 \leq 2k'\kappa$, as needed. □

3.2.6 Zero-Knowledge Variant

Using the "context hiding" technique of [GVW15b], Construction 3.5 for binary functions $f\colon \{0,1\}^k \to \{0,1\}$ can be modified to have zero-knowledge commitments and proofs, but a weaker "target" binding property (see Remark 2.4). We describe this modification and briefly sketch the proofs.

Modified algorithms.

- The Setup algorithm additionally includes a uniformly random $\mathbf{u} \in \mathbb{Z}_q^n$ in the public parameter.
- The Commit algorithm does (statistically) *function-hiding* homomorphic evaluation (which uses a randomized decomposition operation $\mathbf{g}^{-1}$), and retains its random choices for generating proofs.
- The Open algorithm, having computed $\mathbf{S}_{f,x}$ that satisfies Equation (3.6), treats it as a trapdoor in the sense of [MP12] for a combined matrix $\mathbf{C}_{f,x}$, i.e.,

$$\underbrace{[\mathbf{C} - x^t \otimes \mathbf{g}^t \mid \mathbf{C}_f - (1 - f(x))^t \otimes \mathbf{g}^t]}_{\mathbf{C}_{f,x}} \cdot \begin{bmatrix} \mathbf{S}_{f,x} \\ -\mathbf{I}_{n\ell} \end{bmatrix} = (1 - 2f(x))^t \otimes \mathbf{g}^t = \pm 1^t \otimes \mathbf{g}^t.$$

  Using this trapdoor, it randomly samples a Gaussian-distributed integral preimage $\mathbf{s}$ for which $\mathbf{C}_{f,x} \cdot \mathbf{s} = \mathbf{u}$, and outputs $\mathbf{s}$ as the proof.
- The Verify algorithm reconstructs $\mathbf{C}_{f,x}$ using y as the purported value of $f(x)$, and checks that $\mathbf{s}$ is sufficiently short and that $\mathbf{C}_{f,x} \cdot \mathbf{s} = \mathbf{u}$.

Zero knowledge and target binding. The modified scheme has zero-knowledge commitments and proofs: function hiding ensures that $\mathbf{C}_f$ reveals nothing about f (formally, a simulator can return a random "dummy" commitment $\mathbf{C}_f$ instead), while Gaussian preimage sampling ensures that proofs reveal nothing beyond the input-output pairs (formally, a simulator can embed a trapdoor in $\mathbf{C}$ that lets it sample from the desired distribution for any $x, f(x)$; for this it is convenient for x to have one "dummy" coordinate that never changes).

The scheme satisfies *target* evaluation binding (see Remark 2.4) under a suitable SIS assumption. In brief, the reduction is given a random SIS challenge $\mathbf{A} \in \mathbb{Z}_q^{n \times W}$. It gets x^* from the adversary, and supplies public parameter $\mathbf{C} = \mathbf{A} + (x^*)^t \otimes \mathbf{g}^t = \mathbf{A}$ and $\mathbf{u} = \mathbf{A}\mathbf{r}$ for a (secret) uniformly random $\mathbf{r} \leftarrow \{0,1\}^W$. It then gets a function f^* from the adversary, and supplies an honestly generated commitment $\mathbf{C}_{f^*}$ (and its underlying randomness); it also computes the associated short multiplier matrix $\mathbf{S}_{f^*,x^*}$ that satisfies Equation (3.6). Finally, the adversary (if successful) outputs a sufficiently short proof $\mathbf{s}^*$ for $y^* = 1 - f^*(x^*)$, which satisfies

$$[\mathbf{C} - (x^*)^t \otimes \mathbf{g}^t \mid \mathbf{C}_{f^*} - f^*(x^*)^t \otimes \mathbf{g}^t] \cdot \mathbf{s}^* = [\mathbf{A} \mid \mathbf{A}\mathbf{S}_{f^*,x^*}] \cdot \mathbf{s}^* = \mathbf{u}.$$

Hence $\mathbf{A}([\mathbf{I}_W \mid \mathbf{S}_{f^*,x^*}]\mathbf{s}^* - \mathbf{r}) = \mathbf{0}$, yielding an SIS solution—which is nonzero with high probability, since $\mathbf{r}$ has large min-entropy in the adversary's view.

4 Concrete Instantiations

In this section we present and analyze several important instantiations of our functional commitment scheme from Section 3.2. These include commitments to functions of *bounded support* (Section 4.1), which encompass vector commitments (Section 4.1.2), and accumulators (Section 4.1.3); *polynomial commitments* (Section 4.2); and commitments to data with openings that reveal functions thereof (Section 4.3).

Each instantiation of is obtained by (1) defining an appropriate function family that captures the desired functionality, (2) showing how to efficiently implement it homomorphically using Theorem 3.3, and (3) analyzing the resulting norm bounds to set the verification threshold κ appropriately (to ensure correctness). Moreover, for all the special-purpose function families, we show how to implement updates via simple compositions, which means the corresponding commitment schemes are statelessly updateable. Finally, we show that the forms of certain function families enable certain optimizations in their corresponding functional commitment schemes.

At a high level, all of our instantiations for specific kinds of functionalities follow a common template. We first identify a (potentially huge) set of "basis" functions, and show how to express the desired functions relatively simply in terms of this basis, e.g., as linear or low-degree combinations. Plugging this into Construction 3.5 (and using its composition properties) then yields a functional commitment scheme for all functions that have "bounded weight" in terms of the basis. The scheme's running time and associated norm bound is therefore determined mainly by the complexity of the basis functions. Moreover, if there are not too many basis functions (or function inputs), then their commitments (and proofs) can be precomputed, so that the "online" running time is determined by the (low) complexity of the combining operation(s).

4.1 Bounded-Support Commitments

Here we instantiate our general functional commitment scheme (Construction 3.5) for the general class of functions having *bounded support* over a potentially huge domain. As we show below, vector commitments, accumulators, and a generalization of both called *key-value* commitments [BBF19] can be expressed as special cases of bounded-support commitments. Moreover, all these schemes are statelessly updateable, via composition.

Representing bounded-support functions. Let $f\colon \mathcal{X} \to \mathcal{Y}$ be a function for some finite $\mathcal{X}, \mathcal{Y}$, where $\mathcal{Y}$ is without loss of generality an additive group with identity element denoted 0, and let $\operatorname{supp}(f) := \{x \in \mathcal{X} : f(x) \neq 0\} \subseteq \mathcal{X}$ denote the support of f. For each $\bar{x} \in \mathcal{X}$, define the "point function" $\mathsf{Eq}_{\bar{x}}\colon \mathcal{X} \to \{0,1\}$ as

$$\mathsf{Eq}_{\bar{x}}(x) = \begin{cases} 1 & \text{if } x = \bar{x} \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

Then f can be expressed as a linear combination of the $\mathsf{Eq}_{\bar{x}}$ functions, as

$$f(x) = \sum_{\bar{x} \in \mathcal{X}} \mathsf{Eq}_{\bar{x}}(x) \cdot f(\bar{x}) = \sum_{\bar{x} \in S := \operatorname{supp}(f)} \mathsf{Eq}_{\bar{x}}(x) \cdot f(\bar{x}) = L_{\vec{f}_S}(\overrightarrow{\mathsf{Eq}}_S(x)), \tag{4.2}$$

where $\overrightarrow{\mathsf{Eq}}_S(x), \vec{f}_S$ are respectively the vectors of $\mathsf{Eq}_{\bar{x}}(x), f(\bar{x})$ over all $\bar{x} \in S$, and $L_{\vec{f}_S} \colon \{0,1\}^{|S|} \to \mathcal{Y}$ is the linear function that outputs the inner product of its argument with $\vec{f}_S$. In summary, we have expressed f as the composition of $\overrightarrow{\mathsf{Eq}}_S$ and $L_{\vec{f}_S}$.

Therefore, by the linear homomorphisms given in Section 3.1.2 and the composition properties of the generic functional commitment scheme (see Section 3.2.2), we immediately have the following. (A norm bound $\kappa_{\mathsf{Eq}_{\mathcal{X}}}$ is derived below.) Similar correctness lemmas can easily be obtained for richer forms of updates (e.g., post-processing of function outputs), with corresponding norm bounds κ.

Lemma 4.1. *Let $\kappa_{\mathsf{Eq}_{\mathcal{X}}}$ be a norm bound for which the functional commitment scheme from Construction 3.5 is correct for function family $\mathcal{F}_{\mathsf{Eq}_{\mathcal{X}}}$. Then for any $\mathcal{Y} = \mathbb{Z}_q^{n \times d}$, any $s \geq 1$, and any $\kappa \geq s \cdot w \cdot \kappa_{\mathsf{Eq}_{\mathcal{X}}}$, Construction 3.5 is a correct functional commitment scheme for any sum of functions $\mathcal{X} \to \mathcal{Y}$ whose total support size is at most s. The same also holds for $\mathcal{Y} = \{0,1\}$, with the tighter bound $\kappa \geq s \cdot \kappa_{\mathsf{Eq}_{\mathcal{X}}}$.*

Remark 4.2 (Optimizations). Note that $\overrightarrow{\mathsf{Eq}}_S$ depends only on the *support* S of f (not its specific values), and $L_{\vec{f}_S}$ is linear. Together with composition properties of the functional commitment scheme (see Section 3.2.2), these properties allow us to commit to f with a running time, and norm bound, proportional to its support size $|S|$. Moreover, they enable some substantial optimizations. First, if S (or a small enough superset thereof) is known before f itself, then the commitment to $\overrightarrow{\mathsf{Eq}}_S$ can be precomputed, making the "online" commitment to f just a relatively fast linear operation (see Section 3.1.2). Second, and similarly, if a subset $X \subseteq \mathcal{X}$ of potential opening inputs is known in advance, then the openings of $\overrightarrow{\mathsf{Eq}}_S(x)$ for all $x \in X$ can also be precomputed, yielding fast, linear "online" openings of f on these inputs. This precomputation can even be outsourced to an untrusted party, as long as the user verifies the precomputed proofs. Finally, and separately, for functions with binary or small-integer outputs, we can compress commitments and proofs as described in Section 3.2.3.

Instantiating $\mathcal{F}_{\mathsf{Eq}_{\mathcal{X}}}$. We now describe a particularly simple homomorphic implementation of the family $\mathcal{F}_{\mathsf{Eq}_{\mathcal{X}}}$, for which the short multiplier matrices satisfy a small polynomial norm bound.

Let $k \geq \lceil \log_2 |\mathcal{X}| \rceil$ be the length of an element of $\mathcal{X}$, represented as a bit string. Each function $\mathsf{Eq}_{\bar{x}}\colon \{0,1\}^k \to \{0,1\}$ for $\bar{x} \in \mathcal{X}$ can then be implemented (homomorphically) using bit operations, in the following way: on input $x \in \{0,1\}^k$, for each $i \in [k]$ let $e_i = x_i$ if $\bar{x}_i = 1$ and $e_i = 1 - x_i$ if $\bar{x}_i = 0$; this represents whether x_i equals $\bar{x}_i$. Then output the product $\prod_{i \in [k]} e_i$. It is clear that this procedure correctly computes $\mathsf{Eq}_{\bar{x}}(x)$. Its homomorphic implementation consists of a fixed pattern of bit flips, which have no effect on the ultimate norm bound (because $\bar{x}$ is fixed, not an input to the function), followed by a homomorphic product of k bits. So, following Equation (3.5), the ℓ_1 norm bound associated with the homomorphic evaluation of any member of $\mathcal{F}_{\mathsf{Eq}_{\mathcal{X}}}$ is

$$\kappa_{\mathsf{Eq}_{\mathcal{X}}} := (k-1)w + 1 \le kw. \tag{4.3}$$

We remark that the simultaneous (homomorphic) evaluation of *all* $\mathsf{Eq}_{\bar{x}} \in \mathcal{F}_{\mathsf{Eq}_{\mathcal{X}}}$ (i.e., the function $\overrightarrow{\mathsf{Eq}}_{\mathcal{X}}$) can be amortized to save about a $k/2$ factor in the number of (homomorphic) bit multiplications, versus the naïve evaluation of each function individually, which uses $(k-1)\cdot 2^k$ multiplications. On input x, for each $i \in [k]$ we prepare both possible values $x_i, 1 - x_i$ of the bit e_i. Then for $j = k-1, \ldots, 0$, we compute all 2^{k-j} possible partial products $\prod_{i>j} e_i$, by multiplying the previous step's 2^{k-j-1} partial products by the two possible values of e_j.[11] The total number of multiplications used by this method is $4 + 8 + \cdots + 2^k \approx 2^{k+1}$. A similar amortized improvement can be obtained for computing openings at all inputs $x \in \mathcal{X}$, for all the functions.

Discussion. It is instructive to consider the structure of the bounded-support scheme's main intermediate matrices, commitments, and proofs in more detail. As above, let k be the length of the bit-string representation of an element of $\mathcal{X}$. The public parameter is a uniformly random $\mathbf{C} \in \mathbb{Z}_q^{n\times kw}$, where recall that $w = n\ell = n\lceil \log_2 q\rceil$. For every $\bar{x} \in \mathcal{X}$, define the commitment $\mathbf{C}_{\bar{x}} = \mathsf{Commit}(\mathbf{C}, \mathsf{Eq}_{\bar{x}}) \in \mathbb{Z}_q^{n\times w}$; together these define the (potentially enormous) matrix $\mathbf{C}_{\mathcal{X}} = [\mathbf{C}_{\bar{x}}]_{\bar{x}\in\mathcal{X}} \in \mathbb{Z}_q^{n\times|\mathcal{X}|w}$. And for every $x, \bar{x} \in \mathcal{X}$, define the "short" proof[12] $\mathbf{S}_{x,\bar{x}} = \mathsf{Open}(\mathbf{C}, \mathsf{Eq}_{\bar{x}}, x) \in \mathbb{Z}^{kw\times w}$, which satisfies

$$(\mathbf{C} - x^t \otimes \mathbf{g}^t)\cdot \mathbf{S}_{x,\bar{x}} = \mathbf{C}_{\bar{x}} - \mathsf{Eq}_{\bar{x}}(x) \otimes \mathbf{I}_n \otimes \mathbf{g}^t.$$

All this can be represented concisely by the single matrix equation

$$\Big((\mathbf{I}_{|\mathcal{X}|} \otimes \mathbf{C}) - \operatorname{diag}\big(x^t \otimes \mathbf{g}^t\big)_{x\in\mathcal{X}}\Big) \cdot \big(\mathbf{S}_{x,\bar{x}}\big)_{x,\bar{x}\in\mathcal{X}} = \big(\mathbf{1}_{|\mathcal{X}|} \otimes \mathbf{C}_{\mathcal{X}}\big) - \mathbf{I}_{|\mathcal{X}|} \otimes \mathbf{I}_n \otimes \mathbf{g}^t. \tag{4.4}$$

As discussed above in Remark 4.2, any of the matrices $\mathbf{C}_{\bar{x}}, \mathbf{S}_{x,\bar{x}}$ can be precomputed (and verified) in advance, or they can be computed as needed. One can view them as "structured" public parameters for the "online" phase of commitment that depends on the function f.

Now, for a function $f\colon \mathcal{X} \to \mathcal{Y}$, let $\vec{f} = (f(\bar{x}))_{\bar{x}\in\mathcal{X}} \in \mathcal{Y}^{|\mathcal{X}|}$ be its "value vector," whose matrix representation is some $\mathbf{F} \in (\mathbb{Z}_q^{n\times n})^{|\mathcal{X}|} = \mathbb{Z}_q^{|\mathcal{X}|n\times n}$. The commitment to f is simply $\mathbf{C}_f = \mathbf{C}_{\mathcal{X}} \cdot \mathbf{S}_{\times\mathbf{F}}$ for the "short" matrix $\mathbf{S}_{\times\mathbf{F}} = \mathbf{g}^{-1}(\mathbf{F} \otimes \mathbf{g}^t) \in \mathbb{Z}^{|\mathcal{X}|w\times w}$ (see Section 3.1.2). Naturally, any blocks of $\mathbf{C}_{\mathcal{X}}$ corresponding to zero outputs of f can be skipped, because the corresponding blocks of $\mathbf{S}_{\times\mathbf{F}}$ are zero; this enables computing $\mathbf{C}_f$ in time roughly proportional to $|\operatorname{supp}(f)|$. Notice that multiplying the right-hand side of Equation (4.4) by $\mathbf{S}_{\times\mathbf{F}}$ yields $\mathbf{1}_{|\mathcal{X}|} \otimes \mathbf{C}_f - \mathbf{F} \otimes \mathbf{g}^t$, i.e., the xth block is $\mathbf{C}_f$ minus the xth block of $\mathbf{F} \otimes \mathbf{g}^t$, i.e., the robust matrix encoding of $f(x)$. Similarly, the (row) blocks of $(\mathbf{S}_{x,\bar{x}}) \cdot \mathbf{S}_{\times\mathbf{F}}$ are proofs for all inputs $x \in \mathcal{X}$. To verify for a particular x, one just checks these xth blocks against each other using the xth row of the matrix at the left of Equation (4.4), which is possible because that matrix is block diagonal.

4.1.1 Key-Value Commitments

A *key-value map* for a key space $\mathcal{X}$ and a value space $\mathcal{Y}$ is a set of pairs in $\mathcal{X} \times \mathcal{Y}$ whose first entries are mutually distinct, i.e., each key in $\mathcal{X}$ has at most one associated value in $\mathcal{Y}$. In a key-value commitment

[11] We use decreasing j here simply for consistency with the notation in Section 3.1.3, but any order can work.

[12] For convenience of matrix operations, we have swapped the indices of $\mathbf{S}_{x,\bar{x}}$ from the usual $\mathbf{S}_{f,x}$ form.

scheme, a user first commits (concisely) to such a map. Later, the user can (concisely) prove that a given pair $(k, v) \in \mathcal{X} \times \mathcal{Y}$ is in the committed map. The key-binding property says that it is infeasible to prove two different values $v \neq v'$ for the same key. Stateless updateability means that, without needing to know the current contents of the map, the user can add, remove, or change key-value pairs, along with any existing proofs.

Instantiation. We obtain key-value commitments as a special case of bounded-support commitments. We simply represent any key-value map over $\mathcal{X} \times \mathcal{Y}$ as a function $f\colon \mathcal{X} \to \mathcal{Y}$ (and vice-versa) in the natural way, i.e., $f(x) = y$ for each key-value pair (x, y) in the map, and $f(x') = 0$ for all keys $x' \in \mathcal{X}$ that do not have an associated value.[13] Clearly, the support size of f is the number of entries in the map.

We can update (the function representing) a key-value map simply by composing with a suitable update map (function). For example, to insert a key-value pair (k, v) when k does not already have an associated value, or to add v to the existing value for k, we simply add the update function $\delta_{k,v}(x) := v \cdot \mathsf{Eq}_k(x)$. To delete (k, v) from the map, we subtract $\delta_{k,v}(x)$.[14] By Lemma 4.1, this instantiation works for any initial map and sequence of updates having a bounded total number of keys (with multiplicity). In addition, using the composition properties of the functional commitment scheme, we can perform other kinds of updates on the map, like arbitrary post-processing of its values.

4.1.2 Vector Commitments

A vector commitment scheme for d-dimensional vectors over a message space $\mathcal{M}$ is merely a special case of key-value commitment, where $\mathcal{X} = [d]$ and $\mathcal{Y} = \mathcal{M}$. That is, a vector $\mathbf{m} \in \mathcal{M}^d$ corresponds to a key-value map consisting of the pairs (i, m_i) for all $i \in [d]$. (Equivalently, the vector $\mathbf{m}$ corresponds to the function $f_{\mathbf{m}}\colon \mathcal{X} \to \mathcal{Y}$ defined as $f_{\mathbf{m}}(i) = m_i$.) Clearly, the support size of any vector is at most the domain size d. As a special case of key-value commitments, the vector commitment scheme supports all the same stateless update operations on the vector entries. Finally, because the index i corresponds to the key, (selective) key binding is equivalent to (selective) position binding for vector commitments, i.e., it should be infeasible to open a vector commitment to two different values at the same index i.

Precomputation. In vector commitments, the domain size $d = |\mathcal{X}|$ is typically considered to be polynomially bounded, so both optimizations described in Remark 4.2 apply. Specifically, the commitment to $\overrightarrow{\mathsf{Eq}}_{[d]}$ and its openings at every $i \in [d]$ can be precomputed; moreover, this can even be outsourced, as described in Section 3.2.2. Then committing to a vector, and opening it an any position, is just a relatively fast linear combination of these precomputed values. The size of the precomputed data is proportional to $\tilde{O}(d^2)$, which is asymptotically about the same as several prior vector commitments, but here the setup is *untrusted*. The result is a special case of Equation (4.4) with $\mathcal{X} = [d]$.

Comparison to other SIS-based vector commitments. We now briefly compare our vector commitment scheme to prior SIS-based ones [PSTY13, PPS21]. The public parameters for the “base” vector commitment

[13] Note that this makes 0 the implicit ‘default’ value for all such keys. If we wish to have a distinguished ‘undefined’ value $\perp$ for such keys, we can replace $\mathcal{Y}$ with a larger group like $\mathcal{Y}' = \mathcal{Y} \times C$ for some nontrivial cycle C, representing each $y \in \mathcal{Y}$ by $(y, 1) \in \mathcal{Y}'$, and letting $\perp$ be represented by the identity element $(0, 0) \in \mathcal{Y}'$. Note that this encoding requires some care regarding updates, because, for example, representing $y - y = 0$ by $(y, 1) - (y, 1) = (0, 0)$ yields $\perp$, not the encoding of $0 \in \mathcal{Y}$. A simple solution is for all insertions and deletions to use 1s in their second components, and for all modifications of existing values to use 0s.

[14] See Footnote 13 for how insertions/deletions can be handled separately from additions/subtractions, when using an encoding that has a distinguished ‘undefined’ value $\perp$ that is separate from 0.

scheme from [PPS21], for d-dimensional vectors of b-bit messages, consist of: uniformly random $\mathbf{C}_j \in \mathbb{Z}_q^{n \times b}$ for each $j \in [d]$, defining $\mathbf{C}_{[d]} = [\mathbf{C}_1 \mid \cdots \mid \mathbf{C}_d] \in \mathbb{Z}_q^{n \times bd}$; a single uniformly random $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ that for each $i \in [d]$ defines some $\mathbf{A}_i \in \mathbb{Z}_q^{n \times m}$; and for all $i \neq j$, a "short" random matrix $\mathbf{S}_{i,j} \in \mathbb{Z}^{m \times b}$ that satisfies $\mathbf{A}_i \cdot \mathbf{S}_{i,j} = \mathbf{C}_j$. Letting $\mathbf{S}_{i,i} = \mathbf{0}$ for all $i \in [d]$, all this can be represented concisely by the single matrix equation

$$\operatorname{diag}(\mathbf{A}_i)_{i \in [d]} \cdot \big(\mathbf{S}_{i,j}\big)_{i,j \in [d]} = \big(\mathbf{1}_d \otimes \mathbf{C}_{[d]}\big) - \operatorname{diag}(\mathbf{C}_j)_{j \in [d]}. \tag{4.5}$$

This has several similarities to Equation (4.4), but generating these parameters requires a trusted setup that uses secret randomness (including discrete Gaussian sampling to generate the short $\mathbf{S}_{i,j}$), and the size of the parameters grows at least as d^2, which is prohibitive for even moderate dimensions. By contrast, our setup uses only public randomness, and the size of the public parameter grows only as $\operatorname{poly}(\log d)$. Commitments and proofs in [PPS21] are generated and verified very similarly to what is described following Equation (4.4), exploiting the repeated structure of $\mathbf{1}_d \otimes \mathbf{C}_{[d]}$ and the block-diagonal structure of $\operatorname{diag}(\mathbf{A}_i)$.

Finally, the works of [PSTY13, PPS21] also describe specialized Merkle tree-like vector commitments that, unlike ordinary Merkle trees, are statelessly updateable. Asymptotically, our scheme matches or outperforms these in terms of commitment and proof sizes. Moreover, the verifiers from [PSTY13, PPS21] must check a separate short solution to a linear system for *each layer* of their trees, whereas our verifier checks a *single* short solution (of a correspondingly larger dimension). This is a moderate advantage when proving that verification accepts in zero knowledge or with a SNARG, because proving short solutions to linear relations is moderately expensive in these contexts.

4.1.3 Accumulators

A cryptographic *accumulator* [BdM93] is a scheme to concisely commit to a (polynomial-sized) subset of some (possibly huge) universe. This is another special case of key-value commitment, where $\mathcal{X}$ is taken to be the universe and $\mathcal{Y} = \{0, 1\}$. A subset $S \subseteq \mathcal{X}$ corresponds to a key-value map consisting of the pairs $(x, 1)$ for all $x \in S$. (Equivalently, the subset S corresponds to its indicator function $f_S \colon \mathcal{X} \to \{0, 1\}$, defined as $f_S(x) = 1$ if $x \in S$ and $f_S(x) = 0$ otherwise.) Clearly, the number of keys (i.e., support size) is the cardinality of the set. As a special case of key-value commitments, the accumulator supports stateless updates, so elements can be "dynamically" [CL02] added and removed from the committed set. Finally, because keys correspond to universe elements, (selective) key binding is equivalent to set-binding for the accumulator, i.e., it should be infeasible to prove that a value is both in, and not in, the committed set.

4.2 Polynomial Commitments

Here we show how polynomial commitments can be constructed as an instantiation of our general functional commitment scheme, via an analogous approach as for bounded-support commitments, and with similar efficiency and updateability properties. Specifically, we view a polynomial as the composition of a linear function (specified by the coefficients) and a vector of fixed non-linear functions, namely the powers of the input.

Let $\mathcal{R}$ be a finite commutative ring. For an integer $i \geq 0$, define $\mathsf{Pow}_i \colon \mathcal{R} \to \mathcal{R}$ as $\mathsf{Pow}_i(x) = x^i$, and for a positive integer d, define $\overrightarrow{\mathsf{Pow}}_d$ to be the vector of functions Pow_i over $i = 0, \ldots, d-1$. For any $d \geq 1$ with $k = \lceil \log_2 d \rceil$, we can evaluate $\overrightarrow{\mathsf{Pow}}_d$ by evaluating $\overrightarrow{\mathsf{Pow}}_{2^k}$ recursively, using a depth-k tree of at most 2^k ring multiplications.

Univariate polynomials. A univariate polynomial $f(x) = \sum_{i=0}^{d-1} f_i \cdot x^i$ of degree less than d over $\mathcal{R}$ can be expressed as the composition of $\overrightarrow{\mathsf{Pow}}_d$ and the $\mathcal{R}$-linear function $L_{\vec{f}}(\cdot) := \langle \vec{f}, \cdot \rangle$ for the coefficient vector $\vec{f}$ of f, as

$$f(x) = L_{\vec{f}}(\overrightarrow{\mathsf{Pow}}_d(x)).$$

Therefore, we can evaluate f by evaluating $\overrightarrow{\mathsf{Pow}}_d$ in multiplicative depth k, then multiplying the results component-wise with $\vec{f}$, then adding the results via a depth-k binary tree of addition operations.[15]

By the composition properties of the generic functional commitment scheme (see Section 3.2.2), we immediately have the following basic, generic instantiation. Similar correctness lemmas can easily be obtained for richer forms of composition, like updating, adding, multiplying, or dividing polynomials. All of this also generalizes to "sparse" polynomials (i.e., ones with a bounded number of nonzero monomials) of potentially huge degree.

Lemma 4.3. *Let $\kappa_{\mathsf{Mul}}, \kappa_{\mathsf{Add}}$ be norm bounds for which the functional commitment from Construction 3.5 is correct for the multiplication and addition operations in $\mathcal{R}$, respectively. Let $d \geq 1$ be a degree bound and $k = \lceil \log_2 d \rceil$. Then for any $\kappa \geq \kappa_{\mathsf{Mul}}^{k+1} \cdot \kappa_{\mathsf{Add}}^{k}$, Construction 3.5 is a correct polynomial commitment scheme for polynomials over $\mathcal{R}$ of degree less than d.*

For almost all concrete rings of interest, addition and multiplication are not "natively" supported by the homomorphic operations detailed in Section 3.1. Instead, ring operations typically need to be implemented via rich combinations of homomorphic operations on the bit representations of ring elements, and this complexity affects the norm bounds $\kappa_{\mathsf{Mul}}, \kappa_{\mathsf{Add}}$ (as well as the running times of committing and opening). However, for suitable rings like finite fields $\mathbb{F}_{p^{n'}}$, the linear function $L_{\vec{f}}$ *can* be implemented "natively" using just the linear homomorphisms from Section 3.1.2, because the coefficients of f are *known*. (See Remark 4.4 below for further details.)

Remark 4.4 (Optimizations). Similarly to bounded-support commitments (see Remark 4.2), the commitment to $\overrightarrow{\mathsf{Pow}}_d$ can be precomputed, making the "online" commitment to f more efficient. Similarly, if a subset $R \subset \mathcal{R}$ of potential opening inputs is known in advance, then the openings of $\overrightarrow{\mathsf{Pow}}_d$ for $x \in R$ can also be precomputed. However, note that this may not make the online computations linear, because ring addition and multiplication (with a known element) may not correspond to linear homomorphic operations.

For a ring that can be embedded into the matrix ring $\mathbb{Z}_q^{n \times n}$, we can further improve the complexity of the "online" phase of committing and opening, making them just *linear* operations, by "flattening" the Pow_i commitments to use the matrix representations of their outputs. This sacrifices the ability to do further *multiplicative* compositions on the powers—but linear combinations, like those needed to commit to a polynomial, are still supported (because the polynomial coefficients are known).

In brief: suppose that for homomorphic computation, the bit representation $\mathsf{bits}(x)$ of any ring element $x \in \mathcal{R}$ is such that each entry of its matrix representation $\mathbf{R} \in \mathbb{Z}_q^{n \times n}$ is some fixed $\mathbb{Z}_q$-linear function of $\mathsf{bits}(x)$. (If not, we can homomorphically convert to such a representation when it is needed.) Then there exists a matrix $\mathbf{M}$ over $\mathbb{Z}_q$ for which $(\mathsf{bits}(r)^t \otimes \mathbf{I}_n) \cdot \mathbf{M} = \mathbf{R}$. So, given a commitment to any function (e.g., Pow_i) that outputs the bit representation of a ring element, we can "flatten" it to a commitment of the same function, but whose output is represented (robustly) in the matrix ring $\mathbb{Z}_q^{n \times n}$. This is done simply by composing the commitment with the linear function given by $\mathbf{M} \otimes \mathbf{g}^t$, i.e., right-multiplying the commitment by the short matrix $\mathbf{S}_{\times \mathbf{M}} = \mathbf{g}^{-1}(\mathbf{M} \otimes \mathbf{g}^t)$.

[15] More generally, the treatment is essentially the same for polynomials whose coefficients come from some $\mathcal{R}$-module.

Multivariate polynomials. Multivariate polynomials can also be viewed as compositions of Pow functions and a linear function specified by the polynomial's coefficients.

Recall that the *individual degree* of a multivariate polynomial is the maximum degree of any single variable.[16] For a multivariate polynomial with m variables and individual degree less than d, the monomials can be indexed by $D = \{0, \ldots, d-1\}^m$, where each entry of the index is the exponent of the corresponding variable in the monomial. That is, the index $\mathbf{e} \in D$ corresponds to the monomial $\mathsf{Pow}_{\mathbf{e}}(x_1, \ldots, x_m) := x_1^{e_1} \cdots x_m^{e_m}$. Any polynomial $f \colon \mathcal{R}^m \to \mathcal{R}$ on m variables with individual degree less than d can be written as

$$f(x_1, \ldots, x_m) = \sum_{\mathbf{e} \in D} f_{\mathbf{e}} \prod_{i \in [m]} x_i^{e_i} = L_{\vec{f}_S}(\overrightarrow{\mathsf{Pow}}_S(x_1, \ldots, x_m)),$$

where each coefficient $f_{\mathbf{e}} \in \mathcal{R}$ (many of which may be zero), where $S = \operatorname{supp}(f) := \{\mathbf{e} \in D \colon f_{\mathbf{e}} \neq 0\}$ is the *support* of f; $\vec{f}_S, \overrightarrow{\mathsf{Pow}}_S$ are the vectors of $f_{\mathbf{e}}, \mathsf{Pow}_{\mathbf{e}}$ over all $\mathbf{e} \in S$ (respectively); and $L_{\vec{f}_S}(\cdot) := \langle \vec{f}_S, \cdot \rangle$.

We now discuss the computation of the commitment to $\overrightarrow{\mathsf{Pow}}_S$. By definition, each entry of $\overrightarrow{\mathsf{Pow}}_S$ can be written as the product of *univariate* Pow_i functions, i.e., it has the form $\prod_{i \in [m]} \mathsf{Pow}_{e_i}(x_i)$ for some $\mathbf{e} \in S$. So, we can compute a commitment to $\overrightarrow{\mathsf{Pow}}_S$ by multiplicatively composing commitments to the components of $\overrightarrow{\mathsf{Pow}}_d$. Interestingly, we only need to compute commitments to each $\mathsf{Pow}_{\mathbf{e}}$ *up to permutation*, i.e., we can use the same commitment for any $\mathbf{e}'$ that is a permutation of $\mathbf{e}$. This is simply because upon opening, we can permute the input values appropriately.

4.3 Functional Commitments for Bounded Boolean Functions

In this section, we describe how to commit to *input* data and then open to various *functions* of it. This is the notion of functional commitment originally put forth in [LRY16] and also considered in [PPS21]. The core technique is the standard one of swapping "code" and "data" using a universal evaluator. Let $\mathcal{C} := \{C \colon \mathcal{X} \to \mathcal{Y}\}$ be the family of circuits of depth at most D, and let $U \colon \mathcal{C} \times \mathcal{X} \to \mathcal{Y}$ be a depth-universal circuit for this family, for which $U(C, x) = C(x)$; there exists such a circuit of depth $O(D')$ [CH85]. Define $U_x(\cdot) := U(\cdot, x)$. To commit to an input $x \in \mathcal{X}$, use Construction 3.5 to commit to the function U_x. We can then open the commitment for a circuit C (implementing a desired function f) in the usual way, by treating C as the input to the committed function. Following Theorem 3.3, we can use a suitable $\kappa = O(w)^{O(D)}$ for the verification norm bound.

For (constant-width) branching programs of size at most S, we proceed similarly. We first obtain a universal branching program for programs of this size, by applying Barrington's theorem [Bar86] to a certain universal circuit $C_{BP}(B, x)$ that evaluates a given size-S branching program B on a given input x. It can be constructed to have depth $D' = O(\log S)$, because B can be evaluated using a $\lceil \log_2 S \rceil$-depth tree of multiplications of (constant-dimensional) permutation matrices. Applying Barrington's theorem to C_{BP} then gives a (constant-width) universal branching program of size $4^{D'} = \operatorname{poly}(S)$.

Define $C_x(\cdot) := C_{BP}(\cdot, x)$. To commit to an input x, we simply use Construction 3.5 to commit to the function C_x. We can then open the commitment for a size-S branching program B (implementing a desired function f) in the usual way, but treating B as the input to the committed function. Following Theorem 3.3, we can use a suitable $\kappa = w^{O(1)} \cdot \operatorname{poly}(S)$ for the verification norm bound.

[16] Our treatment also adapts straightforwardly to polynomials of bounded *total* degree. We use individual degree because it follows more naturally from the univariate case.

References

[ACL$^+$22] M. R. Albrecht, V. Cini, R. W. F. Lai, G. Malavolta, and S. A. K. Thyagarajan. Lattice-based SNARKs: Publicly verifiable, preprocessing, and recursively composable. In *CRYPTO*, pages 102–132. 2022. Pages 2, 3, 4, and 7.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems. *Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996. Pages 2, 4, and 8.

[AP14] J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In *CRYPTO*, pages 297–314. 2014. Page 10.

[AR20] S. Agrawal and S. Raghuraman. KVaC: Key-value commitments for blockchains and beyond. In *ASIACRYPT*, pages 839–869. 2020. Page 7.

[Bar86] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. Preliminary version in STOC 1986. Page 24.

[BBB$^+$18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, pages 315–334. 2018. Page 3.

[BBC$^+$18] C. Baum, J. Bootle, A. Cerulli, R. del Pino, J. Groth, and V. Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In *CRYPTO*, pages 669–699. 2018. Pages 3 and 7.

[BBF19] D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In *CRYPTO*, pages 561–586. 2019. Pages 7 and 18.

[BC12] N. Bitansky and A. Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *CRYPTO*, pages 255–272. 2012. Page 2.

[BCFL22] D. Balbás, D. Catalano, D. Fiore, and R. W. F. Lai. Functional commitments for circuits from falsifiable assumptions. Cryptology ePrint Archive, Paper 2022/1365, 2022. `https://eprint.iacr.org/2022/1365`. Page 4.

[BDFG21] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. Halo Infinite: Proof-carrying data from additive polynomial commitments. In *CRYPTO*, pages 649–680. 2021. Pages 2 and 3.

[BdM93] J. C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In *EUROCRYPT*, volume 765, pages 274–285. 1993. Pages 2 and 22.

[BFS20] B. Bünz, B. Fisch, and A. Szepieniec. Transparent SNARKs from DARK compilers. In *EUROCRYPT*, pages 677–706. 2020. Pages 2 and 3.

[BGG$^+$14] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556. 2014. Page 10.

[BGV11] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111–131. 2011. Page 2.

[BLS19] J. Bootle, V. Lyubashevsky, and G. Seiler. Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. In *CRYPTO*, pages 176–202. 2019. Page 3.

[BNO21] D. Boneh, W. Nguyen, and A. Ozdemir. Efficient functional commitments: How to commit to a private function. Cryptology ePrint Archive, Paper 2021/1342, 2021. `https://eprint.iacr.org/2021/1342`. Pages 2 and 3.

[BPMW16] F. Bourse, R. D. Pino, M. Minelli, and H. Wee. FHE circuit privacy almost for free. In *CRYPTO*, pages 62–89. 2016. Pages 10 and 13.

[BV14] Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, pages 1–12. 2014. Page 10.

[CEO22] M. Campanelli, F. Engelmann, and C. Orlandi. Zero-knowledge for homomorphic key-value commitments with applications to privacy-preserving ledgers. In *SCN*, pages 761–784. 2022. Page 7.

[CF13] D. Catalano and D. Fiore. Vector commitments and their applications. In *PKC*, pages 55–72. 2013. Pages 2 and 4.

[CFT22] D. Catalano, D. Fiore, and I. Tucker. Additive-homomorphic functional commitments and applications to homomorphic signatures. In *ASIACRYPT*, pages 159–188. 2022. Page 2.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS*, pages 383–395. 1985. Page 2.

[CH85] S. A. Cook and H. J. Hoover. A depth-universal circuit. *SIAM J. Comput.*, 14(4):833–839, 1985. Page 24.

[CJJ21] A. R. Choudhuri, A. Jain, and Z. Jin. SNARGs for P from LWE. In *FOCS*, pages 68–79. 2021. Page 5.

[CL02] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76. 2002. Page 22.

[CPSZ18] A. Chepurnoy, C. Papamanthou, S. Srinivasan, and Y. Zhang. Edrax: A cryptocurrency with stateless transaction validation. Cryptology ePrint Archive, Report 2018/968, 2018. `https://eprint.iacr.org/2018/968`. Pages 2 and 4.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194. 1986. Pages 2 and 3.

[GPV08] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206. 2008. Page 8.

[GSW13] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, pages 75–92. 2013. Pages 2, 5, 6, and 10.

[GSW23] R. Ghosal, A. Sahai, and B. Waters. Non-interactive publicly-verifiable delegation of committed programs. In *PKC*, pages ??–?? 2023. Pages 5 and 9.

[GVW15a] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, pages 503–523. 2015. Page 10.

[GVW15b] S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *STOC*, pages 469–477. 2015. Pages 2, 3, 5, 6, 10, and 17.

[GW11] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108. 2011. Pages 2 and 7.

[IKO07] Y. Ishai, E. Kushilevitz, and R. Ostrovsky. Efficient arguments without short PCPs. In *IEEE Conference on Computational Complexity*, pages 278–291. 2007. Page 2.

[Kil92] J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *STOC*, pages 723–732. 1992. Page 2.

[KLVW23] Y. T. Kalai, A. Lombardi, V. Vaikuntanathan, and D. Wichs. Boosting batch arguments and RAM delegation. In *STOC*, pages ??–?? 2023. Page 5.

[KPV22] A. A. Kattis, K. Panarin, and A. Vlasov. RedShift: Transparent SNARKs from list polynomial commitments. In *CCS*, pages 1725–1737. 2022. Page 3.

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, pages 177–194. 2010. Pages 2 and 3.

[Lee21] J. Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *TCC*, pages 1–34. 2021. Page 3.

[Lis05] M. D. Liskov. Updatable zero-knowledge databases. In *ASIACRYPT*, pages 174–198. 2005. Page 2.

[LM06] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP (2)*, pages 144–155. 2006. Page 14.

[LNP22] V. Lyubashevsky, N. K. Nguyen, and M. Plançon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. In *CRYPTO*, pages 71–101. 2022. Page 3.

[LP20] H. Lipmaa and K. Pavlyk. Succinct functional commitment for a large class of arithmetic circuits. In *ASIACRYPT*, pages 686–716. 2020. Page 2.

[LRY16] B. Libert, S. C. Ramanna, and M. Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In *ICALP*, pages 30:1–30:14. 2016. Pages 2 and 24.

[LY10] B. Libert and M. Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In *TCC*, pages 499–517. 2010. Page 2.

[Mic94] S. Micali. CS proofs. In *FOCS*, pages 436–453. 1994. Page 2.

[Mic02] D. Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007. Preliminary version in FOCS 2002. Page 14.

[MP12] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718. 2012. Pages 5, 10, and 17.

[MR04] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. Preliminary version in FOCS 2004. Pages 4 and 8.

[MRK03] S. Micali, M. O. Rabin, and J. Kilian. Zero-knowledge sets. In *FOCS*, pages 80–91. 2003. Page 2.

[PPS21] C. Peikert, Z. Pepin, and C. Sharp. Vector and functional commitments from lattices. In *TCC*, pages 480–511. 2021. Pages 2, 3, 4, 5, 6, 8, 10, 21, 22, and 24.

[PR06] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC*, pages 145–166. 2006. Page 14.

[PS19] C. Peikert and S. Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In *CRYPTO*, pages 89–114. 2019. Page 10.

[PST13] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In *TCC*, pages 222–242. 2013. Page 2.

[PSTY13] C. Papamanthou, E. Shi, R. Tamassia, and K. Yi. Streaming authenticated data structures. In *EUROCRYPT*, pages 353–370. 2013. Pages 2, 3, 4, 21, and 22.

[Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):1–40, 2009. Preliminary version in STOC 2005. Page 5.

[SBZ01] R. Steinfeld, L. Bull, and Y. Zheng. Content extraction signatures. In *ICISC*, pages 285–304. 2001. Page 2.

[VP19] A. Vlasov and K. Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. Cryptology ePrint Archive, Paper 2019/1020, 2019. `https://eprint.iacr.org/2019/1020`. Page 3.

[WW23] H. Wee and D. Wu. Succinct vector, polynomial, and functional commitments from lattices. In *EUROCRYPT*, pages ??–?? 2023. Pages 4 and 5.