Fine-Grained Secure Computation

Matteo Campanelli and Rosario Gennaro

The City College of New York
matteo.campanelli@gmail.com and rosario@ccny.cuny.edu

Abstract. This paper initiates a study of *Fine Grained Secure Computation*: i.e. the construction of *secure computation primitives* against "moderately complex" adversaries. We present definitions and constructions for Fully Homomorphic Encryption and Verifiable Computation secure against (*non-uniform*) NC^1 adversaries. We also present two application scenarios for our model: *(i)* hardware chips that prove their own correctness, and *(ii)* protocols against rational adversaries potentially relevant to the *Verifier's Dilemma* in smart-contracts transactions such as Ethereum.

1 Introduction

Historically Cryptography has been used to protect information (either in transit or stored) from unauthorized access. One of the most important developments in Cryptography in the last thirty years, has been the ability to protect non only information but also the *computations* that are performed on data that needs to be secure. Starting with the work on secure multiparty computation [Yao82], and continuing with ZK proofs [GMR89], and more recently Fully Homomorphic Encryption [Gen09], verifiable outsourcing computation [GKR08,GGP10], SNARKs [GGPR13,BCI$^+$13] and obfuscation [GGH$^+$16] we now have cryptographic tools that protect the secrecy and integrity not only of data, but also of the programs which run on that data.

Another crucial development in Modern Cryptography has been the adoption of a more "fine-grained" notion of computational hardness and security. The traditional cryptographic approach modeled computational tasks as "easy" (for the honest parties to perform) and "hard" (infeasible for the adversary). Yet we have also seen a notion of *moderately hard* problems being used to attain certain security properties. The best example of this approach might be the use of moderately hard inversion problems used in blockchain protocols such as Bitcoin. Although present in many works since the inception of Modern Cryptography, this approach was first formalized in a work of Dwork and Naor [DN92].

In this paper we consider the following model (which can be traced back to the seminal paper by Merkle [Mer78] on public key cryptography). Honest parties will run a protocol which will cost[1] them C while an adversary who

[1] We intentionally refer to it as "cost" to keep the notion generic. For concreteness one can think of C as the running time required to run the protocol.

wants to compromise the security of the protocol will incur a $C' = \omega(C)$ cost. Note that while C' is asymptotically larger than C, it might still be a feasible cost to incur – the only guarantee is that it is substantially larger than the work of the honest parties. For example in Merkle's original proposal for public-key cryptography the honest parties can exchange a key in time T but the adversary can only learn the key in time T^2. Other examples include primitives introduced by Cachin and Maurer [CM97] and Hastad [Has87] where the cost is the space and parallel time complexity of the parties, respectively.

Recently there has been renewed interest in this model. Degwekar et al. [DVV16] show how to construct certain cryptographic primitives in NC^1 [resp. $\mathsf{A}C^0$] which are secure against all adversaries in NC^1 [resp. $\mathsf{A}C^0$]. In conceptually related work Ball et al. [BRSV17] present computational problems which are "moderately hard" on average, if they are moderately hard in the worst case, a useful property for such problems to be used as cryptographic primitives.

The goal of this paper is to initiate a study of *Fine Grained Secure Computation*. By doing so we connect these two major developments in Modern Cryptography. The question we ask is if it is possible to construct *secure computation primitives* that are secure against "moderately complex" adversaries. We answer this question in the affirmative, by presenting definitions and constructions for the task of Fully Homomorphic Encryption and Verifiable Computation in the fine-grained model. We also present two application scenarios for our model: i) hardware chips that prove their own correctness and ii) protocols against rational adversaries including potential solutions to the *Verifier's Dilemma* in smart-contracts transactions such as Ethereum.

1.1 Our Results

Our starting point is the work in [DVV16] and specifically their public-key encryption scheme secure against NC^1 circuits. Recall that $\mathsf{AC}^0[2]$ is the class of Boolean circuits with constant depth, unbounded fan-in, augmented with parity gates. If the number of AND gates of non constant fan-in is constant we say that the circuit belongs to the class $\mathsf{AC}^0_{\mathrm{Q}}[2] \subset \mathsf{AC}^0[2]$.

Our results can be summarized as follows

- We first show that the techniques in [DVV16] can be used to build a somewhat homomorphic encryption (SHE) scheme. We note that because honest parties are limited to NC^1 computations, the best we can hope is to have a scheme that is homomorphic for computations in NC^1. However our scheme can only support computations that can be expressed in $\mathsf{AC}^0_{\mathrm{Q}}[2]$.
- We then use our SHE scheme, in conjunction with protocols described in [GGP10,CKV10,AIK10], to construct verifiable computation protocols for functions in $\mathsf{AC}^0_{\mathrm{Q}}[2]$, secure and input/output private against any adversary in NC^1.

Our somewhat homomorphic encryption also allows us to obtain the following protocols secure against NC^1 adversaries: *(i)* constant-round 2PC, secure in the presence of semi-honest static adversaries for functions in $\mathsf{AC}^0_{\mathsf{Q}}[2]$; *(ii) Private Function Evaluation* in a two party setting for circuits of constant *multiplicative* depth without relying on universal circuits. These results stem from well-known folklore transformations and we do not prove them formally.

The class $\mathsf{AC}^0_{\mathsf{Q}}[2]$ includes many natural and interesting problems such as: fixed precision arithmetic, evaluation of formulas in 3CNF (or kCNF for any constant k), a representative subset of SQL queries, and S-Boxes [BP11] for symmetric key encryption.

Our results (like [DVV16]) hold under the assumption that $\oplus\mathsf{L/poly} \subsetneq \mathsf{NC}^1$, a widely believed worst-case assumption on separation of complexity classes. Notice that this assumption does not imply the existence of one-way functions (or even $\mathsf{P} \neq \mathsf{NP}$). Thus, our work shows that it is possible to obtain "advanced" cryptographic schemes[2] such as somewhat homomorphic encryption and verifiable computation even if we do not live in Minicrypt[3].

Comparison with other approaches. One important question is: on what features are our schemes better than "generic" cryptographic schemes that after all are secure against *any* polynomial time adversary.

One such feature is the type of assumption one must make to prove security. As we said above, our schemes rely on a very mild worst-case complexity assumption, while cryptographic SHE and VC schemes rely on very specific assumptions, which are much stronger than the above.

For the case of Verifiable Computation, we also have information-theoretic protocols which are secure against *any* (possibly computationally unbounded) adversary. For example the "Muggles" protocol in [GKR08] which can compute any (log-space uniform) NC function, and is also reasonably efficient in practice [CMT12]. Or, the more recent work [GR18], which obtains efficient VC for functions in a subset of $\mathsf{NC} \cap \mathsf{SC}$. Compared to these results, one aspect in which our protocol fares better is that our Verifier can be implemented with a constant-depth circuit (in particular in TC^0, see Section 4) which is not possible for the Verifier in [GKR08,GR18]. Moreover our protocol is non-interactive (while [GKR08,GR18] requires logarithmically many rounds of interaction) and because our protocols work in the "pre-processing model" we do not require any uniformity or regularity condition on the circuit being outsourced (which are required by [GKR08] and [CMT12]).

Interactive proofs (again, we stress, with information-theoretic soundness) with verification in constant depth are discussed in [GGH$^+$07] (where the verifier is in NC^0). We point out that our schemes besides achieving non-interactive constant-depth verification also has a verifier that runs in linear *sequential* time on the input/output size (i.e. in $O(\lambda^c(n+m))$ where λ is the security parameter, n the input and m the output sizes of the function being outsourced).

[2] Naturally the security guarantees of these schemes are more limited compared to their standard definitions.

[3] This is a reference to Impagliazzo's "five possible worlds" [Imp95].

1.2 Overview of our Techniques

In [DVV16] the authors already point out that their scheme is linearly homomorphic. We make use of the *re-linearization* technique from [BV14] to construct a leveled homomorphic encryption.

Our scheme (as the one in [DVV16]) is secure against adversaries in the class of (*non-uniform*) NC^1. This implies that we can only evaluate functions in NC^1 otherwise the evaluator would be able to break the semantic security of the scheme. However we have to ensure that the *whole* homomorphic evaluation stays in NC^1. The problem is that homomorphically evaluating a function f might increase the depth of the computation.

In terms of circuit depth, the main overhead will be (as usual) the computation of multiplication gates. As we show in Section 3 a single homomorphic multiplication can be performed by a depth two $\mathsf{AC}^0[2]$ circuit, but this requires depth $O(\log(n))$ with a circuit of fan-in two. Therefore, a circuit for f with $\omega(1)$ multiplicative depth would require an evaluation of $\omega(\log(n))$ depth, which would be out of NC^1. Therefore our first scheme can only evaluate functions with constant multiplicative depth, as in that case the evaluation stays in $\mathsf{AC}^0[2]$.

We then present a second scheme that extends the class of computable functions to $\mathsf{AC}^0_{\mathrm{Q}}[2]$ by allowing for a negligible error in the correctness of the scheme. We use a result by Razborov [Raz87] on approximating $\mathsf{AC}^0[2]$ circuits with low-degree polynomials – the correctness of the approximation (appropriately amplified) will be the correctness of our scheme.

1.3 Application Scenarios

The applications described in this section refer to the problem of Verifying Computation, where a Client outsources an algorithm f and an input x to a Server, who returns a value y and a proof that $y = f(x)$. The security property is that it should be infeasible to convince the verifier to accept $y' \neq f(x)$, and the crucial efficiency property is that verifying the proof should cost less than computing f (since avoiding that cost was the reason the Client hired the Server to compute f).

HARDWARE CHIPS THAT PROVE THEIR OWN CORRECTNESS Verifiable Computation (VC) can be used to verify the execution of hardware chips designed by untrusted manufacturers. One could envision chips that provide (efficient) *proofs of their correctness* for every input-output computation they perform. These proofs must be *efficiently verified* in less time and energy than it takes to re-execute the computation itself.

When working in hardware, however, one may not need the full power of cryptographic protection against *any* malicious attacks since one could bound the computational power of the malicious chip. The bound could be obtained by making (reasonable and evidence-based) assumptions on how much computational power can fit in a given chip area. For example one could safely assume that a malicious chip can perform at most a constant factor more work than the original function because of the basic physics of the size and power constraints. In other words, if C is the cost of the honest Server in a VC protocol, then in this model the adversary is limited to $O(C)$-cost computations, and therefore a protocol that guarantees that succesful cheating strategies require $\omega(C)$ cost, will suffice. This is exactly the model in our paper. Our results will apply to the case in which we define the cost as the depth (i.e. the parallel time complexity) of the computation implemented in the chip.

Rational Proofs. The problem above is related to the notion of composable Rational Proofs defined in [CG15]. In a Rational Proof (introduced by Azar and Micali [AM12,AM13]), given a function f and an input x, the Server returns the value $y = f(x)$, and (possibly) some auxiliary information, to the Client. The Client in turn pays the Server for its work with a reward based on the transcript exchanged with the server and some randomness chosen by the client. The crucial property is that this reward is maximized in expectation when the server returns the correct value y. Clearly a *rational* prover who is only interested in maximizing his reward, will always answer correctly.

The authors of [CG15] show however that the definition of Rational Proofs in [AM12,AM13] does not satisfy a basic compositional property needed for the case in which many computations are outsourced to many servers who compete with each other for rewards (e.g. the case of volunteer computations [ACK+02]). A "rational proof" for the single-proof setting may no longer be rational when a large number of "computation problems" are outsourced. If one can produce T "random guesses" to problems in the time it takes to solve 1 problem correctly, it may be preferable to guess! That's because even if each individual reward for an incorrect answer is lower than the reward for a correct answer, the total reward of T incorrect answers might be higher (and this is indeed the case for some of the protocols presented in [AM12,AM13]).

The question (only partially answered in [CG15,CG17] for a limited class of computations) is to design protocols where the reward is strictly connected, not just to the correctness of the result, but to the amount of work done by the prover. Consider for example a protocol where the prover collects the reward only if he produces a proof of correctness of the result. Assume that the cost to produce a valid proof for an incorrect result, is higher than just computing the correct result and the correct proof. Then obviously a rational prover will always answer correctly, because the above strategy of fast incorrect answers will not work anymore.

While the application is different, the goal is the same as in the previous verifiable hardware scenario.

The Verifier's Dilemma. In blockchain systems such as Ethereum, transactions can be expressed by arbitrary programs. To add a transaction to a block miners have to verify its validity, which could be too costly if the program is too complex. This creates the so-called *Verifier's Dilemma* [LTKS15]: given a costly valid transaction Tr a miner who spends time verifying it is at a disadvantage over a miner who does not verify it and accept it "uncritically" since the latter will produce a valid block faster and claim the reward. On the other hand if the transaction is invalid, accepting it without verifying it first will lead to the rejection of the entire block by the blockchain and a waste of work by the uncritical miner. The solution is to require efficiently verifiable proofs of validity for transactions, an approach already pursued by various startups in the Ethereum ecosystem (e.g. TrueBit[4]). We note that it suffices for these proofs to satisfy the condition above: i.e. we do not need the full power of information-theoretic or cryptographic security but it is enough to guarantee that to produce a proof of correctness for a false transaction is more costly than producing a valid transaction and its correct proof, which is exactly the model we are proposing.

1.4 Future Directions

Our work opens up many interesting future directions.

First of all, it would be nice to extend our results to the case where cost is the actual running time, rather than "parallel running time"/"circuit depth" as in our model. The techniques in [BRSV17] (which presents problems conjectured to have $\Omega(n^2)$ complexity on the average), if not even the original work of Merkle [Mer78], might be useful in building a verifiable computation scheme where if computing the function takes time T, then producing a false proof of correctness would have to take $\Omega(T^2)$.

For the specifics of our constructions it would be nice to "close the gap" between what we can achieve and the complexity assumption: our schemes can only compute $\mathsf{AC}^0_{\mathsf{Q}}[2]$ against adversaries in NC^1, and ideally we would like to be able to compute all of NC^1 (or at the very least all of $\mathsf{AC}^0[2]$).

Finally, to apply these schemes in practice it is important to have tight concrete security reductions and a proof-of-concept implementations.

2 Preliminaries

For a distribution D, we denote by $x \leftarrow D$ the fact that x is being sample according to D. We remind the reader that an ensemble $\mathcal{X} = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ is a family of probability distributions over a family of domains $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$. We say two ensembles $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}' = \{D'_\lambda\}_{\lambda\in\mathbb{N}}$ are statistically indistinguishable if $\frac{1}{2}\sum_x |D(x) - D'(x)| < \mathsf{neg}(\lambda)$. Finally, we note that all arithmetic computations (such as sums, inner product, matrix products, etc.) in this work will be over GF(2) unless specified otherwise.

Definition 2.1 (Function Family). *A* function family *is a family of (possibly randomized) functions $F = \{f_\lambda\}_{\lambda\in\mathbb{N}}$, where for each λ, f_λ has domain D^f_λ and co-domain R^f_λ. A* class *$\mathcal{C}$ is a collection of function families.*

In most of our constructions $D^f_\lambda = \{0,1\}^{d^f_\lambda}$ and $R^f_\lambda = \{0,1\}^{r^f_\lambda}$ for sequences $\{d^f_\lambda\}_\lambda$, $\{d^f_\lambda\}_\lambda$.

[4] TrueBit: *https://truebit.io/*

In the rest of the paper we will focus on the class of $\mathcal{C} = \mathsf{NC}^1$ of functions for which there is a polynomial $p(\cdot)$ and a constant c such that for each λ, the function f_λ can be computed by a Boolean (randomized) fan-in 2, circuit of size $p(\lambda)$ and depth $c\log(\lambda)$. In the formal statements of our results we will also use the following classes: AC^0, the class of functions of polynomial size and constant depth with $\mathsf{AND}, \mathsf{OR}$ and NOT gates with unbounded fan-in; $\mathsf{AC}^0[2]$, the class of functions of polynomial size and constant depth with $\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}$ and PARITY gates with unbounded fan-in; TC^0, the class of functions of polynomial size and constant depth with $\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}$ and $\mathsf{MAJORITY}$ gates with unbounded fan-in. Given a function f, we define *multiplicative depth* of f as the degree of the lowest-degree polynomial in GF(2) that evaluates to f.

Limited Adversaries. We define adversaries also as families of randomized algorithms $\{A_\lambda\}_\lambda$, one for each security parameter (note that this is a non-uniform notion of security). We denote the class of adversaries we consider as $\mathcal{A}$, and in the rest of the paper we will also restrict $\mathcal{A}$ to NC^1.

Infinitely-Often Security. We now move to define security against all adversaries $\{A_\lambda\}_\lambda$ that belong to a class $\mathcal{A}$.

Our results achieve an "infinitely often" notion of security, which states that for all adversaries outside of our permitted class $\mathcal{A}$ our security property holds infinitely often (i.e. for an infinite sequence of security parameters rather than for every sufficiently large security parameter. We inherit this limitation from the techniques of [DVV16].

Definition 2.2 (Infinitely-Often Computational Indistinguishability). *Let $\mathcal{X} = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ Let $\mathcal{Y} = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ be ensembles over the same domain family, $\mathcal{A}$ a class of adversaries, and Λ an infinite subset of $\mathbb{N}$. We say that $\mathcal{X}$ and $\mathcal{Y}$ are infinitely often computational indistinguishable with respect to set Λ and the class $\mathcal{A}$, denoted by $\mathcal{X} \sim_{\Lambda,\mathcal{A}} \mathcal{Y}$ if there exists a negligible function ν such that for any $\lambda \in \Lambda$ and for any adversary $A = \{A_\lambda\}_\lambda \in \mathcal{A}$*

$$|\Pr[A_\lambda(X_\lambda) = 1] - \Pr[A_\lambda(Y_\lambda) = 1]| < \nu(\lambda)$$

When $\mathcal{A} = \mathsf{NC}^1$ we will keep it implicit and use the notation $\mathcal{X} \sim_\Lambda \mathcal{Y}$ and say that $\mathcal{X}$ and $\mathcal{Y}$ are Λ-computationally indistinguishable.

In our proofs we will use the following facts on infinitely-often computationally indistinguishable ensembles. We skip their proof as, except for a few technicalities, it is analogous to the corresponding properties for standard computational indistinguishability[5].

Lemma 2.1 (Facts on Λ-Computational Indistinguishability).

- ***Transitivity:*** *Let $m = \mathsf{poly}(\lambda)$ and $\mathcal{X}^{(j)}$ with $j \in \{0, \ldots, m\}$ be ensembles. If for all $j \in [m]$ $\mathcal{X}^{(j-1)} \sim_\Lambda \mathcal{X}^{(j)}$, then $\mathcal{X}^{(0)} \sim_\Lambda \mathcal{X}^{(m)}$.*
- ***Weaker than statistical indistinguishability:*** *Let $\mathcal{X}, \mathcal{Y}$ be statistically indistinguishable ensembles. Then $\mathcal{X} \sim_\Lambda \mathcal{Y}$ for any infinite $\Lambda \subseteq \mathbb{N}$*
- ***Closure under NC^1:*** *Let $\mathcal{X}, \mathcal{Y}$ be ensembles and $\{f_\lambda\}_{\lambda\in\mathbb{N}} \in \mathsf{NC}^1$. If $\mathcal{X} \sim_\Lambda \mathcal{Y}$ for some Λ then $f_\lambda(\mathcal{X}) \sim_\Lambda f_\lambda(\mathcal{Y})$.*

[5] We refer the reader to [Gol01].

2.1 Public-Key Encryption

A public-key encryption scheme
$\mathsf{PKE} = (\mathsf{PKE.Keygen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ is a triple of algorithms which operate as follow:

- **Key Generation.** The algorithm $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Keygen}(1^\lambda)$ takes a unary representation of the security parameter and outputs a public key encryption key pk and a secret decryption key sk.
- **Encryption.** The algorithm $c \leftarrow \mathsf{PKE.Enc}_{\mathsf{pk}}(\mu)$ takes the public key pk and a single bit message $\mu \in \{0, 1\}$ and outputs a ciphertext c. The notation $\mathsf{PKE.Enc}_{\mathsf{pk}}(\mu; r)$ will be used to represent the encryption of a bit μ using randomness r.
- **Decryption.** The algorithm $\mu^* \leftarrow \mathsf{PKE.Dec}_{\mathsf{sk}}(c)$ takes the secret key sk and a ciphertext c and outputs a message $\mu^* \in \{0, 1\}$.

Obviously we require that $\mu = \mathsf{PKE.Dec}_{\mathsf{sk}}(\mathsf{PKE.Enc}_{\mathsf{pk}}(\mu))$

Definition 2.3 (CPA Security for PKE). *A scheme* PKE *is IND-CPA secure if for an infinite* $\Lambda \subseteq \mathbb{N}$ *we have*

$$(\mathsf{pk}, \mathsf{PKE.Enc}_{\mathsf{pk}}(0)) \sim_{\Lambda} (\mathsf{pk}, \mathsf{PKE.Enc}_{\mathsf{pk}}(1))$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Keygen}(1^\lambda)$*.*

Remark 2.1 (Security for Multiple Messages). Notice that by a standard hybrid argument and Lemma 2.1 we can prove that any scheme secure according to Definition 2.3 is also secure for multiple messages (i.e. the two sequences of encryptions bit by bit of two bit strings are computationally indistinguishable). We will use this fact in the proofs in Section 4, but we do not provide the formal definition for this type of security. We refer the reader to 5.4.2 in [Gol09].

Somewhat Homomorphic Encryption A public-key encryption scheme is said to be homomorphic if there is an additional algorithm Eval which takes a input the public key pk, the representation of a function $f : \{0, 1\}^l \to \{0, 1\}$ and a set of l ciphertexts $c_1, \ldots, c_l$, and outputs a ciphertext c_f.

We proceed to define the homomorphism property. The next notion of $\mathcal{C}$-homomorphism is sometimes also referred to as "somewhat homomorphism".

Definition 2.4 ($\mathcal{C}$-homomorphism). *Let* $\mathcal{C}$ *be a class of functions (together with their respective representations). An encryption scheme* PKE *is* $\mathcal{C}$*-homomorphic (or, homomorphic for the class* $\mathcal{C}$*) if for every function* f_λ *where* $f_\lambda \in \mathcal{F}\{f_\lambda\}_{\lambda \in \mathbb{N}} \in \mathcal{C}$ *and respective inputs* $\mu_1, \ldots, \mu_l \in \{0, 1\}$ *(where* $l = l(\lambda)$*), it holds that if* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Keygen}(1^\lambda)$ *and* $c_i \leftarrow \mathsf{PKE.Enc}_{\mathsf{pk}}(\mu_i)$ *then*

$$\Pr[\mathsf{PKE.Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}(F, c_1, \ldots, c_l)) \neq F(\mu_1, \ldots, \mu_l)] = \mathsf{neg}(\lambda),$$

As usual we require the scheme to be non-trivial by requiring that the output of Eval is compact:

Definition 2.5 (Compactness). *A homomorphic encryption scheme* PKE *is compact if there exists a polynomial* s *in* λ *such that the output length of* Eval *is at most* $s(\lambda)$ *bits long (regardless of the function* f *being computed or the number of inputs).*

Definition 2.6. *Let* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *of arithmetic circuits in* $GF(2)$*. A scheme* PKE *is leveled* $\mathcal{C}$*-homomorphic if it takes* 1^L *as additional input in key generation, and can only evaluate depth-*L *arithmetic circuits from* $\mathcal{C}$*. The bound* $s(\lambda)$ *on the ciphertext must remain independent of* L*.*

2.2 Verifiable Computation

In a *Verifiable Computation* scheme a Client uses an untrusted server to compute a function f over an input x. The goal is to prevent the Client from accepting an incorrect value $y' \neq f(x)$. We require that the Client's cost of running this protocol be smaller than the cost of computing the function on his own. The following definition is from [GGP10] which allows the client to run a possibly expensive pre-processing step.

Definition 2.7 (Verifiable Computation Scheme).
A verifiable computation scheme $\mathcal{VC} = (\mathsf{VC.KeyGen}, \mathsf{VC.ProbGen}, \mathsf{VC.Compute}, \mathsf{VC.Verify})$ *consists of the four algorithms defined below.*

1. $\mathsf{VC.KeyGen}(f, 1^\lambda) \rightarrow (\mathsf{pk_W}, \mathsf{sk_D})$*: Based on the security parameter* λ*, the randomized* key generation *algorithm generates a public key that encodes the target function* F*, which is used by the Server to compute* F*. It also computes a matching secret key, which is kept private by the Client.*
2. $\mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x) \rightarrow (q_x, s_x)$*: The* problem generation *algorithm uses the secret key* $\mathsf{sk_D}$ *to encode the function input* x *as a public query* q_x *which is given to the Server to compute with, and a secret value* s_x *which is kept private by the Client.*
3. $\mathsf{VC.Compute}_{\mathsf{pk_W}}(q_x) \rightarrow a_x$*: Using the Client's public key and the encoded input, the Server* computes *an encoded version of the function's output* $y = F(x)$*.*
4. $\mathsf{VC.Verify}_{\mathsf{sk_D}}(s_x, a_x) \rightarrow y\ \cup \{\perp\}$*: Using the secret key* $\mathsf{sk_D}$ *and the secret "decoding"* s_x*, the* verification *algorithm converts the worker's encoded output into the output of the function, e.g.,* $y = f(x)$ *or outputs* $\perp$ *indicating that* a_x *does not represent the valid output of* F *on* x*.*

The scheme should be complete, i.e. an honest Server should (almost) always return the correct value.

Definition 2.8. *Completeness A delegation scheme* $\mathcal{VC} = (\mathsf{VC.KeyGen}, \mathsf{VC.ProbGen}, \mathsf{VC.Compute}, \mathsf{VC.Verify})$ *has overwhelming completeness for a class of functions* $\mathcal{C}$ *if there is a function* $\nu(n) = \mathsf{neg}(\lambda)$ *such that for infinitely many values of* λ*, if* $f_\lambda \in \mathcal{F} \in \mathcal{C}$*, then for all inputs* x *the following holds with probability at least* $1 - \nu(n)$*:* $(\mathsf{pk_W}, \mathsf{sk_D}) \leftarrow \mathsf{VC.KeyGen}(f_\lambda, \lambda)$ $(q_x, s_x) \leftarrow \mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x)$ *and* $a_x \leftarrow \mathsf{VC.Compute}_{\mathsf{pk_W}}(q_x)$ *then* $y = f_\lambda(x) \leftarrow \mathsf{VC.Verify}_{\mathsf{sk_D}}(s_x, a_x)$*.*

To define soundness we consider an adversary who plays the role of a malicious Server who tries to convince the Client of an incorrect output $y \neq f(x)$. The adversary is allowed to run the protocol on inputs of her choice, i.e. see the *queries* q_{x_i} for adversarially chosen x_i's before picking an input x and attempt to cheat on that input. Because we are interested in the parallel complexity of the adversary we distinguish between two parameters l and m. The adversary is allowed to do l rounds of adaptive queries, and in each round she queries m inputs. Jumping ahead, because our adversaries are restricted to NC^1 circuits, we will have to bound l with a constant, but we will be able to keep m polynomially large.

Experiment $\mathbf{Exp}_A^{\mathsf{Verif}}[\mathcal{VC}, f, \lambda, l, m]$
  $(\mathsf{pk_W}, \mathsf{sk_D}) \leftarrow \mathsf{VC.KeyGen}(f, \lambda)$;
  $\mathcal{I} \leftarrow \emptyset$;
  For $i = 1, \dots, i = l$;
    $\{x_{(i-1)m}, \dots x_{im-1}\} \leftarrow A_\lambda(\mathsf{pk_W}, \mathcal{I})$;
    $\{(q_j, s_j) : (q_j, s_j) \leftarrow \mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x_j), j \in \{(i-1)m, \dots, im\}\}$
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{x_{(i-1)m}, \dots x_{im-1}\} \cup \{q_{(i-1)m}, \dots q_{im-1}\}$;
  $\hat{a} \leftarrow A_\lambda(\mathsf{pk_W}, \mathcal{I})$;
  $\hat{y} \leftarrow \mathsf{VC.Verify}_{\mathsf{sk_D}}(s_{lm}, \hat{a})$
  If $\hat{y} \neq \perp$ and $\hat{y} \neq f(x_{lm})$, output 1, else 0.

Remark 2.2. In the experiment above the adversary "tries to cheat" on the last input presented in the last round of queries (i.e. $x_l m$). This is without loss of generality. In fact, assume the adversary aimed at cheating on an input presented before round l, then with one additional round it could present that same input once more as the last of the batch in that round.

Definition 2.9 (Soundness). *We say that a verifiable computation scheme is (l, m)-sound against a class $\mathcal{A}$ of adversaries if there exists a negligible function $\mathsf{neg}(\lambda)$, such that for all $A = \{A_\lambda\}_\lambda \in \mathcal{A}$, and for infinitely many λ we have that*

$$\Pr[\mathbf{Exp}_A^{\mathsf{Verif}}[\mathcal{VC}, f, \lambda, l, m] = 1] \leq \mathsf{neg}(\lambda)$$

Assume the function f we are trying to compute belongs to a class $\mathcal{C}$ which is smaller than $\mathcal{A}$. Then our definition guarantees that the "cost" of cheating is higher than the cost of honestly computing f and engaging in the Verifiable Computation protocol $\mathcal{VC}$. Jumping ahead, our scheme will allow us to compute the class $\mathcal{C} = \mathsf{AC}^0[2]$ against the class of adversaries $\mathcal{A} = \mathsf{NC}^1$.

EFFICIENCY The last thing to consider is the efficiency of a VC protocol. Here we focus on the time complexity of computing the function f. Let n be the number of input bits, and m be the number of output bits, and S be the size of the circuit computing f.

- A verifiable computation scheme $\mathcal{VC}$ is **client-efficient** if circuit sizes of $\mathsf{VC.ProbGen}$ and $\mathsf{VC.Verify}$ are $o(S)$. We say that it is **linear-client** if those sizes are $O(\mathsf{poly}(\lambda)(n + m))$.

– A verifiable computation scheme $\mathcal{VC}$ is **server-efficient** if the circuit size of $\mathsf{VC.Compute}$ is $O(\mathsf{poly}(\lambda)S)$.

We note that the key generation protocol $\mathsf{VC.KeyGen}$ can be expensive, and indeed in our protocol (as in [GGP10,CKV10,AIK10]) its cost is the same as computing f – this is OK as $\mathsf{VC.KeyGen}$ is only invoked once per function, and the cost can be amortized over several computations of f.

3 Fine-Grained SHE

We start by recalling the public key encryption from [DVV16] which is secure against adversaries in NC^1.

The scheme is described in Figure 1. Its security relies on the following result, implicit in [IK00][6]. We will also use this lemma when proving the security of our construction in Section 3.

Lemma 3.1 ([IK00]). *If* $\oplus\mathsf{L/poly} \subsetneq \mathsf{NC}^1$ *then there exist distribution* $\mathcal{D}^{kg}_\lambda$ *over* $\{0,1\}^{\lambda\times\lambda}$, *distribution* $\mathcal{D}^{f}_\lambda$ *over matrices in* $\{0,1\}^{\lambda\times\lambda}$ *of full rank, and infinite set* $\Lambda \subseteq \mathbb{N}$ *such that*

$$\mathbf{M}^{kg} \sim_\Lambda \mathbf{M}^f$$

where $\mathbf{M}^f \leftarrow \mathcal{D}^f_\lambda$ *and* $\mathbf{M}^{kg} \leftarrow \mathcal{D}^{kg}_\lambda$.

The following result is central to the correctness of the scheme PKE in Figure 1 and is implicit in [DVV16].

Lemma 3.2 ([DVV16]). *There exists sampling algorithm* $\mathsf{KSample}$ *such that* $(\mathbf{M}, \mathbf{k}) \leftarrow \mathsf{KSample}(1^\lambda)$, $\mathbf{M}$ *is a matrix distributed according to* $\mathcal{D}^{kg}_\lambda$ *(as in Lemma 3.1),* $\mathbf{k}$ *is a vector in the kernel of* $\mathbf{M}$ *and has the form* $\mathbf{k} = (r_1, r_2, \ldots, r_{\lambda-1}, 1) \in \{0,1\}^\lambda$ *where* r_i*-s are uniformly distributed bits.*

Theorem 3.1 ([DVV16]). *Assume* $\oplus\mathsf{L/poly} \subsetneq \mathsf{NC}^1$. *Then, the scheme* $\mathsf{PKE} = (\mathsf{PKE.Keygen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *defined in Figure 1 is a Public Key Encryption scheme secure against* NC^1 *adversaries. All algorithms in the scheme are computable in* $\mathsf{AC}^0[2]$.

3.1 Leveled Homomorphic Encryption for $\mathsf{AC}^0_\mathsf{Q}[2]$ Functions Secure against NC^1

We denote by $\mathbf{x}[i]$ the i-th bit of a vector of bits $\mathbf{x}$. Below, the scheme $\mathsf{PKE} = (\mathsf{PKE.Keygen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ is the one defined in Figure 1.

Our SHE scheme is defined by the following four algorithms:

[6] Stated as Lemma 4.3 in [DVV16].

– $\mathsf{PKE.Keygen}_{\mathsf{sk}}(1^\lambda)$:
  1. Sample $(\mathbf{M}, \mathbf{k}) \leftarrow \mathsf{KSample}(1^\lambda)$;
  2. Output $(\mathsf{pk} = \mathbf{M}, \mathsf{sk} = \mathbf{k})$.
– $\mathsf{PKE.Enc}_{\mathsf{pk}=\mathbf{M}}(\mu))$:
  1. Sample $\mathbf{r} \leftarrow_\$ \{0,1\}^\lambda$;
  2. Let $t^\intercal = (0 \ \ldots 0 \ 1) \in \{0,1\}^\lambda$;
  3. Output $\mathbf{c}^\intercal = \mathbf{r}^\intercal \mathbf{M} + \mu \mathbf{t}^\intercal$.
– $\mathsf{PKE.Dec}_{\mathsf{sk}=\mathbf{k}}(\mathbf{c})$:
  1. Output $\langle \mathbf{k} \, , \mathbf{c} \rangle$

Fig. 1. PKE construction [DVV16]

– $\mathsf{HE.Keygen}_{\mathsf{sk}}(1^\lambda, L)$: For key generation, sample $L+1$ key pairs $(\mathbf{M}_0, \mathbf{k}_0), \ldots, \mathbf{M}_0, \mathbf{k}_L) \leftarrow \mathsf{PKE.Keygen}(1^\lambda)$, and compute, for all $\ell \in \{0, \ldots, L-1\}$, $i, j \in [\lambda]$, the value

$$\mathbf{a}_{\ell,i,j} \leftarrow \mathsf{PKE.Enc}_{\mathbf{M}_{\ell+1}}(\mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \in \{0,1\}^\lambda$$

  We define $\mathbf{A} := \{a_{\ell,i,j}\}_{\ell,i,j}$ to be the set of all these values. t then outputs the secret key $\mathsf{sk} = \mathbf{k}_L$, and the public key $\mathsf{pk} = (\mathbf{M}_0, \mathbf{A})$. In the following we call $\mathsf{evk} = \mathbf{A}$ the evaluation key.

  We point out a property that will be useful later: by the definition above, for all $\ell \in \{0, \ldots, L-1\}$ we have

$$\langle \mathbf{k}_{\ell+1} \, , \mathbf{a}_{\ell+1,i,j} \rangle = \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j] \, . \tag{1}$$

– $\mathsf{HE.Enc}_{\mathsf{pk}}(\mu))$: Recall that $\mathsf{pk} = \mathbf{M}_0$. To encrypt a message μ we compute $\mathbf{v} \leftarrow \mathsf{PKE.Enc}_{\mathbf{M}_0}(\mu)$. The output ciphertext contains $\mathbf{v}$ in addition to a "level tag", an index in $\{0, \ldots, L\}$ denoting the "multiplicative depth" of the generated ciphertext. The encryption algorithm outputs $c := (\mathbf{v}, 0)$.
– $\mathsf{HE.Dec}_{\mathbf{k}_L}(c)$: To decrypt a ciphertext[7] $c = (\mathbf{v}, L)$ compute $\mathsf{PKE.Dec}_{\mathbf{k}_L}(\mathbf{v})$, i.e.

$$\langle \mathbf{k}_L \, , \mathbf{v} \rangle$$

– $\mathsf{HE.Eval}_{\mathsf{evk}}(f, c_1, \ldots, c_t)$: where $F : \{0,1\}^t \to \{0,1\}$: We require that f is represented as an arithmetic circuit in GF(2) with addition gates of unbounded fan-in and multiplication gates of fan-in 2. We also require the circuit to be *layered*, i.e. the set of gates can be partitioned in subsets (layers) such that wires are always between adjacent layers. Each layer should be composed homogeneously either of addition or multiplication gates. Finally, we require that the number of multiplications layers (i.e. the multiplicative depth) of f is L.

  We homomorphically evaluate f gate by gate. We will show how to perform multiplication (resp. addition) of two (resp. many) ciphertexts. Carrying out

[7] We are only requiring to decrypt ciphertexts that are output by $\mathsf{HE.Eval}(\cdots)$

this procedure recursively, we can homomorphically compute any circuit f of multiplicative depth L.

Ciphertext structure during evaluation. During the homomorphic evaluation a ciphertext will be of the form $c = (\mathbf{v}, \ell)$ where ℓ is the "level tag" mentioned above. At any point of the evaluation we will have that ℓ is between 0 (for fresh ciphertexts at the input layer) and L (at the output layer). We define homomorphic evaluation only among ciphertexts at the same level. Since our circuit is layered we will not have to worry about homomorphic evaluation occurring among ciphertexts at different levels. Consistently with the fact a level tag represents the multiplicative depth of a ciphertext, addition gates will keep the level of ciphertexts unchanged, whereas multiplication gates will increase it by one. Finally, we will keep the invariant that the output of each gate evaluation $c = (\mathbf{v}, \ell)$ is such that

$$\langle \mathbf{k}_\ell \, , \mathbf{v} \rangle = \mu \tag{2}$$

where μ is the correct plaintext output of the gate.

Homomorphic Evaluation of gates:

- *Addition gates.* Homomorphic evaluation of an addition gates on inputs $c_1, \ldots, c_t$ where $c_i = (\mathbf{v}_i, \ell)$ is performed by outputting

  $$c_{\text{add}} = (\mathbf{v}_{\text{add}}, \ell) := \Big(\sum_i \mathbf{v}_i, \ell\Big)$$

  Informally, one can see that

  $$\langle \mathbf{k}_\ell \, , \mathbf{v}_{\text{add}} \rangle = \langle \mathbf{k}_\ell \, , \sum_i \mathbf{v}_i \rangle = \sum_i \langle \mathbf{k}_\ell \, , \mathbf{v}_i \rangle = \sum_i \mu_i$$

  where μ_i is the plaintext corresponding to $\mathbf{v}_i$. This satisfies the invariant in Eq. 2.
- *Multiplication gates.* We show how to multiply ciphertexts c, c' where $c = (\mathbf{v}, \ell)$ and $c' = (\mathbf{v}', \ell)$ to obtain an output ciphertext $c_{\text{mult}} = (\mathbf{v}_{\text{mult}}, \ell{+}1)$. The homomorphic multiplication algorithm will set

  $$\mathbf{v}_{\text{mult}} := \sum_{i,j \in [\lambda]} h_{i,j} \cdot \mathbf{a}_{\ell+1,i,j}$$

  where $h_{i,j} = \mathbf{v}[i] \cdot \mathbf{v}'[j]$ for $i, j \in [\lambda]$.
  The final output ciphertext will be

  $$c_{\text{mult}} := (\mathbf{v}_{\text{mult}}, \ell + 1).$$

This satisfies the invariant in Eq. 2 as

$$
\begin{aligned}
\langle \mathbf{k}_{\ell+1} \, , \mathbf{v}_{\text{mult}} \rangle &= \langle \mathbf{k}_{\ell+1} \, , \sum_{i,j \in [\lambda]} h_{i,j} \cdot \mathbf{a}_{\ell+1,i,j} \rangle \\
&= \sum_{i,j \in [\lambda]} (h_{i,j} \cdot \langle \mathbf{k}_{\ell+1} \, , \mathbf{a}_{\ell+1,i,j} \rangle) \\
&= \sum_{i,j \in [\lambda]} (h_{i,j} \cdot \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \\
&= \sum_{i,j \in [\lambda]} (\mathbf{v}[i] \cdot \mathbf{v}'[j] \cdot \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \\
&= \Big(\sum_{i \in [\lambda]} \mathbf{v}[i] \cdot \mathbf{k}_\ell[i] \Big) \cdot \Big(\sum_{j \in [\lambda]} \mathbf{v}'[j] \cdot \mathbf{k}_\ell[j] \Big) \\
&= \langle \mathbf{k}_\ell \, , \mathbf{v} \rangle \cdot \langle \mathbf{k}_\ell \, , \mathbf{v}' \rangle \\
&= \mu \cdot \mu'
\end{aligned}
$$

where in the third and fourth equality we used respectively Eq. 1 and the definition of $h_{i,j}$, and μ, μ' are the plaintexts corresponding to $\mathbf{v}$ $\mathbf{v}'$ respectively.

3.2 Security Analysis

Theorem 3.2 (Security). *The scheme* HE *is CPA secure against* NC^1 *adversaries (Definition 2.3) under the assumption* $\oplus\mathsf{L}/\mathsf{poly} \subsetneq \mathsf{NC}^1$.

Proof. We are going to prove that there exists infinite $\Lambda \subseteq \mathbb{N}$ such that $(\mathsf{pk}, \mathsf{evk}, \mathsf{HE.Enc}_{\mathsf{pk}}(0)) \sim_\Lambda (\mathsf{pk}, \mathsf{evk}, \mathsf{HE.Enc}_{\mathsf{pk}}(1))$.

When using the notations $\mathbf{M}^{\text{kg}}$ and $\mathbf{M}^{\text{f}}$ we will always denote matrices to respectively distributed according to $\mathcal{D}^{\text{f}}_\lambda$ and $\mathcal{D}^{\text{kg}}$, where $\mathcal{D}^{\text{f}}_\lambda$ and $\mathcal{D}^{\text{kg}}$ are the distributions defined in Lemma 3.1.

We will define the (randomized) encoding procedure $\mathsf{E} : \{0,1\}^{\lambda \times \lambda} \to \{0,1\}\lambda$ defined as

$$\mathsf{E}(\mathbf{M}, b) = \mathbf{r}^\intercal \mathbf{M} + (0 \ \ldots 0 \ b)^\intercal \, ,$$

where r is uniformly distributed in $\{0,1\}^\lambda$. The functions we will pass to E will be distributed either according to $\mathbf{M}^{\text{kg}}$ or $\mathbf{M}^{\text{f}}$. Notice that: *(i)* $\mathsf{E}(\mathbf{M}^{\text{kg}}, b)$ is distributed identically to $\mathsf{HE.Enc}_{\mathsf{pk}}(b)$; *(ii)* $\mathsf{E}(\mathbf{M}^{\text{f}}, b)$ corresponds to the uniform distribution over $\{0,1\}^\lambda$ because (by Lemma 3.1) $\mathbf{M}^{\text{f}}$ has full rank and hence $\mathbf{r}^\intercal \mathbf{M}^{\text{f}}$ must be uniformly random.

We will denote with $\mathbf{M}^{\text{kg}}_1, \ldots, \mathbf{M}^{\text{kg}}_L$ the matrices $\mathbf{M}_1, \ldots, \mathbf{M}_\ell$ used to construct the evaluation key in $\mathsf{HE.Keygen}$ (see definition). Recall these matrices are distributed according to $\mathcal{D}^{\text{kg}}$ as in Lemma 3.1.

We will also define the following vectors:

$$\alpha^{\text{kg}}_\ell := \{ \mathsf{E}(\mathbf{M}^{\text{kg}}_{\ell+1}, \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \mid i,j \in [\lambda] \} \qquad \alpha^{\text{f}}_\ell := \{ \mathsf{E}(\mathbf{M}^{\text{f}}_{\ell+1}, \mathbf{k}_\ell[i] \cdot \mathbf{k}_\ell[j]) \mid i,j \in [\lambda] \} \, ,$$

where $\mathbf{k}_\ell$ is defined as in $\mathsf{HE.Keygen}$ and the matrices in input to E will be clear from the context. Notice that all the elements of $\alpha_\ell^{\mathrm{kg}}$ are encryptions, whereas all the elements of α_ℓ^{f} are uniformly distributed.

We will use a standard hybrid argument. Each of our hybrids is parametrized by a bit b. This bit informally marks whether the hybrid contains an element indistinguishable from an encryption of b.

- $\mathcal{E}^b := (\mathbf{M}_0^{\mathrm{kg}}, \mathsf{E}(\mathbf{M}_0^{\mathrm{kg}}, b), \alpha_1^{\mathrm{kg}}, \ldots, \alpha_L^{\mathrm{kg}})$ where $\mathbf{M}_0^{\mathrm{kg}}$ corresponds to the public key of our scheme. Notice that $\alpha_\ell^{\mathrm{kg}} \equiv \{\mathbf{a}_{\ell,i,j} \mid i,j \in [\lambda]\}$ where $\mathbf{a}_{\ell,i,j}$ is as defined in $\mathsf{HE.Keygen}$. This hybrid corresponds to the distribution $(\mathsf{pk}, \mathsf{evk}, \mathsf{HE.Enc}_{\mathsf{pk}}(b))$.
- $\mathcal{H}_0^b := (\mathbf{M}_0^{\mathrm{f}}, \mathsf{E}(\mathbf{M}^{\mathrm{f}}, b), \alpha_1^{\mathrm{kg}}, \ldots, \alpha_L^{\mathrm{kg}})$. The only difference from $\mathcal{E}$ is in the first two components where we replaced the actual public key and ciphertext with a full rank matrix distributed according to $\mathcal{D}_\lambda^{\mathrm{f}}$ and a random vector of bits.
- For $\ell \in [L]$ we define

$$\mathcal{H}_\ell^b := (\mathbf{M}_0^{\mathrm{f}}, \mathsf{E}(\mathbf{M}^{\mathrm{f}}, b), \alpha_1^{\mathrm{f}}, \ldots, \alpha_\ell^{\mathrm{f}}, \alpha_{\ell+1}^{\mathrm{kg}}, \ldots, \alpha_L^{\mathrm{kg}})\,.$$

We will proceed proving that

$$\mathcal{E}^0 \sim_\Lambda \mathcal{H}_0^0 \sim_\Lambda \mathcal{H}_1^0 \sim_\Lambda \ldots \sim_\Lambda \mathcal{H}_L^0 \sim_\Lambda \mathcal{H}_L^1 \sim_\Lambda \ldots \sim_\Lambda \mathcal{H}_1^1 \sim_\Lambda \mathcal{H}_0^1 \sim_\Lambda \mathcal{E}^1$$

through a series of smaller claims. In the remainder of the proof Λ refers to the set in Lemma 3.1.

- $\mathcal{E}^0 \sim_\Lambda \mathcal{H}_0^0$: if this were not the case we would be able to distinguish $\mathbf{M}_0^{\mathrm{kg}}$ from $\mathbf{M}_0^{\mathrm{f}}$ for some of the values in the set Λ thus contradicting Lemma 3.1.
- $\mathcal{H}_{\ell-1}^0 \sim_\Lambda \mathcal{H}_\ell^0$ for $\ell \in [L]$: assume by contradiction this statement is false for some $\ell \in [L]$. That is

$$(\mathbf{M}_0^{\mathrm{f}}, \mathsf{E}(\mathbf{M}_0^{\mathrm{f}}, b), \alpha_1^{\mathrm{f}}, \ldots, \alpha_{\ell-1}^{\mathrm{f}}, \alpha_\ell^{\mathrm{kg}}, \ldots, \alpha_L^{\mathrm{kg}}) \not\sim_\Lambda (\mathbf{M}_0^{\mathrm{f}}, \mathsf{E}(\mathbf{M}_0^{\mathrm{f}}, b), \alpha_1^{\mathrm{f}}, \ldots, \alpha_\ell^{\mathrm{f}}, \alpha_{\ell+1}^{\mathrm{kg}}, \ldots, \alpha_L^{\mathrm{kg}})\,.$$

  Recall that, by definition, the elements of $\alpha_\ell^{\mathrm{kg}}$ are all encryptions whereas the elements of α_ℓ^{f} are all randomly distributed values. This contradicts the the semantic security of the scheme PKE (by a standard hybrid argument on the number of ciphertexts).
- $\mathcal{H}_L^0 \sim_\Lambda \mathcal{H}_L^1$: the distributions associated to these two hybrids are identical. In fact, notice the only difference between these two hybrids is in the second component: $\mathsf{E}(\mathbf{M}^{\mathrm{f}}, 0)$ in $\mathcal{H}_L^0$ and $\mathsf{E}(\mathbf{M}^{\mathrm{f}}, 1)$ in $\mathcal{H}_L^1$. As observed above $\mathsf{E}(\mathbf{M}^{\mathrm{f}}, b)$ is uniformly distributed, which proves the claim.

All the claims above can be proven analogously for $\mathcal{E}^1, \mathcal{H}_0^1$ and $\mathcal{H}_\ell^1$-s. □

3.3 Efficiency and Homomorphic Properties of Our Scheme

Our scheme is secure against adversaries in the class NC^1. This implies that we can run $\mathsf{HE.Eval}$ only on functions f that are in NC^1, otherwise the evaluator would be able to break the semantic security of the scheme. However we have to ensure that the *whole* homomorphic evaluation stays in NC^1. The problem is that homomorphically evaluating f has an overhead with respect to the "plain" evaluation of f. Therefore, we need to determine for which functions f, we can guarantee that $\mathsf{HE.Eval}(F, \dots)$ will stay in NC^1.

In terms of circuit depth, the main overhead when evaluating f homomorphically is given by the multiplication gates (addition, on the other hand, is "for free" — see definition of HE.Eval above). A single homomorphic multiplication can be performed by a depth two $\mathsf{AC}^0[2]$ circuit, but this requires depth $\Omega(\log(n))$ with a circuit of fan-in two. Therefore, a circuit for f with $\omega(1)$ multiplicative depth would require an evaluation of $\omega(\log(n))$ depth, which would be out of NC^1. On the other hand, observe that for any function f in $\mathsf{AC}^0[2]$ with constant multiplicative depth, the evaluation stays in $\mathsf{AC}^0[2]$. This because there is a constant number (depth) of homomorphic multiplications each requiring an $\mathsf{AC}^0[2]$ computation.

We can now state the following result, derived from the observations above and the fact that the invariant in Eq. 2 is preserved throughout homomorphic evaluation.

Theorem 3.3. *Let* $\mathsf{AC}^0_{CM}[2]$ *the family of circuits in* $\mathsf{AC}^0[2]$ *with constant multiplicative depth. The scheme* HE *is leveled* $\mathsf{AC}^0_{CM}[2]$*-homomorphic. Key generation, encryption, decryption and evaluation are all computable in* $\mathsf{AC}^0_{CM}[2]$*.*

3.4 Beyond Constant Multiplicative Depth

In the previous section we saw how our scheme is homomorphic for a class of constant-depth, unbounded fan-in arithmetic circuits in GF(2) with *constant multiplicative depth*, i.e. polynomials in GF(2) of constant degree. We now show how to overcome this limitation by slightly changing our scheme and using a result (implicit in [Raz87]) on approximating $\mathsf{AC}^0[2]$ circuits with low-degree polynomials.

Definition 3.1 (Quasi-Constant Multiplicative Depth). *Let* $C \in \mathsf{AC}^0[2]$ *be a circuit. Let* S *be the number of* AND *gates of non constant fan-in. If* $S = O(1)$ *we say that* C *has quasi-constant multiplicative depth. We denote with* $\mathsf{AC}^0_Q[2]$ *the class of circuits with such property.*

Lemma 3.3 ([Raz87]). *Let* C *be an* $\mathsf{AC}^0_Q[2]$ *circuit of depth* d*. Then there exists a randomized circuit* $C' \in \mathsf{AC}^0_{CM}[2]$ *such that, for all* x*,*

$$\Pr[C'(x) \neq C(x)] \leq \epsilon\,,$$

where $\epsilon = O(1)$*. The circuit* C' *uses* $O(n)$ *random bits and its representation can be computed in* NC^1 *from a representation of* C*.*

Below is a variation of our homomorphic scheme that can evaluate all circuits in $\mathsf{AC}^0_{\mathrm{Q}}[2]$ in NC^1. This time, in order to evaluate circuit C, we perform several homomorphic evaluations of the randomized circuit C' (as in Lemma 3.3). To obtain the plaintext output of C we can decrypt all the ciphertext outputs and take the majority result. Notice that this scheme is still compact. As we use a randomized approach to evaluate F, the scheme HE' will be implicitly parametrized by a soundness parameter s. Intuitively, the probability of a function F being evaluated incorrectly will be upper bounded by 2^{-s}.

We define the following auxiliary functions for our scheme:

Definition 3.2 (Auxiliary Functions for HE').
Let $f : \{0,1\}^t \to \{0,1\}$ be represented as an arithmetic circuit as in HE *and* pk *a public key for the scheme* HE *that includes the evaluation key. Let s be a soundness parameter. We denote by f' the randomized function approximating f as in Lemma 3.3; let $t' = O(t)$ be the number of additional random bits f' will take in input.*

– $\mathsf{GenApproxFun}(f)$:
  1. *Computes and returns the representation of the approximating function f'.*
– $\mathsf{SampleAuxRandomness}_s(\mathsf{pk}, f')$:
  1. *We assume f' is the randomized function approximating f as in Lemma 3.3; let $t' = O(t)$ be the number of additional random bits f' will take in input.*
  2. *Sample $s \cdot t'$ random bits $r_1^{(1)}, \ldots, r_{t'}^{(1)}, \ldots, r_1^{(s)}, \ldots, r_{t'}^{(s)}$;*
  3. *Compute $\hat{\mathbf{r}}_{aux} := \{\hat{r}_j^{(i)} \mid \hat{r}_j^{(i)} \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(r_j^{(i)}), i \in [s], j \in [t']\}$;*
  4. *Output $\hat{\mathbf{r}}_{aux}$.*

– $\mathsf{EvalApprox}_s(\mathsf{pk}, f', c_1, \ldots, c_t, \hat{\mathbf{r}}_{aux})$:
  1. *Let $\hat{\mathbf{r}}_{aux} = \{\hat{r}_j^{(i)} \mid i \in [s], j \in [t']\}$.*
  2. *For $i \in [s]$, compute $c_i^{out} \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}(f', c_1, \ldots c_t, \hat{r}_1^{(i)}, \ldots, \hat{r}_{t'}^{(i)})$.*
  3. *Output $\mathbf{c} = (c_1^{out}, \ldots, c_s^{out})$*

The new scheme HE' with soundness parameter s follows.

– Key generation and encryption are the same as in HE.
– $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(f, c_1, \ldots, c_t)$:
  1. Compute $f' \leftarrow \mathsf{GenApproxFun}(f)$;
  2. Compute $\hat{\mathbf{r}}_{\mathrm{aux}} \leftarrow \mathsf{SampleAuxRandomness}_s(\mathsf{pk}, f')$;
  3. Compute $\mathbf{c} \leftarrow \mathsf{EvalApprox}_s(\mathsf{pk}, f', c_1, \ldots, c_t, \hat{\mathbf{r}}_{\mathrm{aux}})$;
  4. Output $\mathbf{c} = (c_1^{\mathrm{out}}, \ldots, c_s^{\mathrm{out}})$.
– $\mathsf{HE}'.\mathsf{Dec}_{\mathsf{sk}}(\mathbf{c})$:
  1. Decrypt all c_i^{out}-s in $\mathbf{c}$ and output the majority bit.

Remark 3.1. Given in input a function f not necessarily of constant multiplicative depth, $\mathsf{GenApproxFun}$ returns a function f' of constant multiplicative depth that approximates it. As stated in Lemma 3.3, $\mathsf{GenApproxFun}$ is computable in uniform NC^1. Notice that this is the only component of $\mathsf{HE}'.\mathsf{Eval}$ that is not computable in $\mathsf{AC}^0_{\mathrm{CM}}[2]$. In fact, $\mathsf{SampleAuxRandomness}$ is clearly in $\mathsf{AC}^0_{\mathrm{CM}}[2]$ and $\mathsf{EvalApprox}$ makes parallel invocations to $\mathsf{HE}.\mathsf{Eval}$ which is computable in $\mathsf{AC}^0_{\mathrm{CM}}[2]$ when provided in input a function in $\mathsf{AC}^0_{\mathrm{CM}}[2]$ (Theorem 3.3). This fact will be useful when showing the completeness of our verifiable computation schemes in Section 4.

Theorem 3.4. *Let $\mathsf{AC}^0_Q[2]$ the family of circuits in $\mathsf{AC}^0[2]$ with quasi-constant multiplicative depth as in Definition 3.1. The scheme HE' above with soundness parameter $s = \Omega(\lambda)$ is leveled $\mathsf{AC}^0_Q[2]$-homomorphic. Key generation and encryption can be computed in $\mathsf{AC}^0[2]$. Evaluation is computable in (uniform) NC^1. Decryption is computable in $\mathsf{AC}^0[2]$ with a single, unbounded fan-in majority gate at the root.*

4 Fine-Grained Verifiable Computation

In this section we describe our private verifiable computation scheme. Our constructions are heavily based on the techniques in [CKV10] to obtain (reusable) verifiable computation from fully homomorphic encryption. In order to guarantee that these techniques also work within NC^1 we prove that: *(i)* the constructions can be computed in low-depth; *(ii)* the reductions in the security proofs can be carried out in low-depth.

THE SCHEME FROM [CKV10]. To derive Verifiable Computation from Homomorphic Encryption, [CKV10] follows this approach. The Client, in the expensive preprocessing phase, selects a random input r, encrypts it $c_r = E(r)$ and homomorphically compute $c_{f(r)}$ an encryption of $f(r)$. During the online phase, the Client, on input x, computes $c_x = E(x)$ and submits the ciphertexts c_x, c_r in random order to the Server, who homomorphically compute $c_{f(r)} = E(f(r))$ and $c_{f(x)} = E(f(x))$ and returns them to the Client. The Client given the message c_0, c_1 from the Server, checks that $c_b = c_{f(r)}$ (for the appropriate bit b) and if so accepts $y = D(c_{f(x)})$ as $y = f(x)$. The semantic security of E guarantees that this protocol has soundness error $1/2$ (which can be reduced by parallel repetition). This scheme is however one-time, as a malicious server can figure out which one is the test ciphertext $c_{f(r)}$ if it is used again.

To make this scheme "many time secure", [CKV10] uses the paradigm introduced in [GGP10] of running the 1-time scheme "under the covers" of a different homomorphic encryption key each time.

4.1 A One-time Verification Scheme

Before we present our variant of the one-time construction in [CKV10], we present two auxiliary lemmas that guarantee that our protocols are computable in $\mathsf{AC}^0[2]$. We refer the reader to [Hag91,MV91] for the proof Lemma 4.1.

Lemma 4.1. *[Hag91,MV91] There are uniform* AC^0 *circuits* $C : \{0,1\}^{\mathsf{poly}(l)} \to [l]^l$ *of size* $\mathsf{poly}(l)$ *and depth* $O(1)$ *whose output distribution have statistical distance* $\leq 2^{-l}$ *from the uniform distribution over permutations of* $[l]$*.*

Lemma 4.2. *There are uniform* $\mathsf{AC}^0[2]$ *circuits* $C : [l]^l \times \{0,1\}^l \to \{0,1\}^l$ *of size* $O(l^2)$ *where* $C(\pi, (x_1, \dots, x_l)) = (\pi(1), \dots, \pi(l))$ *and* π *is a permutation.*

Proof. Let $\mathbf{x} = (x_1, \dots, x_l)$ the bits to permute and let π be a permutation If π is represented as a permutation matrix with rows $\mathbf{r}_1, \dots, \mathbf{r}_l$, we can permute $\mathbf{x}$ by simply performing l parallel inner products $\langle \mathbf{x}\,, \mathbf{r}_i \rangle$-s, which is in $\mathsf{AC}^0[2]$. We now describe how to generate the permutation matrix from a binary representations $x_1, \dots, x_{\mathsf{lg}(l)}$ of the integers in $[l]$. Let $f_i : \{0,1\}^{\mathsf{lg}(l)} \to \{0,1\}^l$ be the function that computes the i-th row of the permutation matrix. We can define f_i as follows:

$$f_i(x_1, \dots, x_{\mathsf{lg}(l)}) := \mathsf{eq}([i-1]_2, (x_1, \dots, x_{\mathsf{lg}(l)})\,,$$

where $[i-1]_2$ is the binary representation of $i-1$ and eq returns 1 if its two inputs (each of lenght $\mathsf{lg}(l)$) are equal. The function f_i is clearly in $\mathsf{AC}^0[2]$. $\square$

The following is an adaptation of the one-time secure delegation scheme from [CKV10]. We make non-black box use of our homomorphic encryption scheme HE' (Section 3.4) with soundness parameter $s = \lambda$. Notice that. during the preprocessing phase, we fix the "auxiliary randomness" for $\mathsf{EvalApprox}$ (and thus for $\mathsf{HE}'.\mathsf{Eval}$) once and for all. We will use that same randomness for all the input instances. This choice does not affect the security of the construction. We remind the reader that we will simplify notation by considering the evaluation key of our somewhat homomorphic encryption scheme as part of its public key.

If x is a vector of bits $x_1, \leftarrow, x_n$, below we will denote with $\mathsf{HE}'.\mathsf{Enc}(x)$ the concatenation of the bit by bit ciphertexts $\mathsf{HE}'.\mathsf{Enc}(x_1), \dots, \mathsf{HE}'.\mathsf{Enc}(x_n)$. We denote by $\mathsf{HE}'.\mathsf{Enc}(\bar{0})$ the concatenation of n encryptions of 0, $\mathsf{HE}'.\mathsf{Enc}(0)$.

Remark 4.1 (On deterministic homomorphic evaluation). As pointed out in [CKV10], one requirement for the approach above to work is for the homomorphic evaluation to be deterministic. We point out that once $\hat{\mathbf{r}}_{\text{aux}}$ are fixed once and for all the homomorphic evaluation in $\mathsf{VC}.\mathsf{Compute}$ is deterministic.

Remark 4.2 (On including f' in pk_W). In the construction above we included f' in the public key lengthening the size of the key. We point out this is not necessary and that f' can be computed by the worker on her own during the execution of $\mathsf{VC}.\mathsf{Compute}$. However this would not allow us to simply homomorphically evaluate $\mathsf{VC}.\mathsf{Compute}$ in the definition of $\overline{\mathcal{VC}}$ in Section 4.2. This because the complexity of $\mathsf{VC}.\mathsf{Compute}$ would go from $\mathsf{AC}^0_{\mathrm{CM}}[2]$ to NC^1, which our homomorphic schemes cannot handle. We point out that it would still be possible to modify the construction of $\overline{\mathcal{VC}}$ not including f' in pk_W to obtain the same completeness and soundness properties. However this would come at a cost of a more complex transformation in Figure 3. Including f' in pk_W allowed us to kept the transformation as simple and close to the original description in [CKV10] as possible.

Let $f : \{0,1\}^n \to \{0,1\}^m$ be a function and $\mathsf{GenApproxFun}, \mathsf{SampleAuxRandomness}$ and $\mathsf{EvalApprox}$ as in Definition 3.2.

- $\mathsf{VC.KeyGen}(1^\lambda, f) \to (\mathsf{pk_W}, \mathsf{sk_D})$: We assume function F represented as
  1. Generate a pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE}'.\mathsf{Keygen}(1^\lambda)$.
  2. Generate the approximating function $f' \leftarrow \mathsf{GenApproxFun}(f)$;
  3. Generate the ciphertext of the auxiliary random input for homomorphic evaluation $\hat{\mathbf{r}}_{\text{aux}} \leftarrow \mathsf{SampleAuxRandomness}_\lambda(\mathsf{pk}, f')$
  4. Compute t independent encryptions $\hat{r}_i = \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}}(\bar{0})$ and the homomorphic evaluations $\hat{w}_i = \hat{f}(\hat{r}_i) = \mathsf{EvalApprox}_s(\mathsf{pk}, f', \hat{r}_i, \hat{\mathbf{r}}_{\text{aux}})$ for $i \in [t]$.
  5. $\mathsf{pk_W} \leftarrow (\mathsf{pk}, f', \hat{\mathbf{r}}_{\text{aux}}), \mathsf{sk_D} \leftarrow \{(\hat{r}_i, \hat{w}_i)_{i\in[t]}\}$.
- $\mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x) \to (q_x, s_x)$:
  1. Compute t independent encryptions $\hat{r}_{i+t} = \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}}(x)$ for $i \in [t]$.
  2. Sample a random permutation $\pi \leftarrow_\$ S_{2t}$.
  3. $q_x \leftarrow (\hat{z}_{\pi(1)}, \ldots, \hat{z}_{\pi(2t)}) = (\hat{r}_1, \ldots, \hat{r}_{2t})$; $s_x \leftarrow \pi$
- $\mathsf{VC.Compute}_{\mathsf{pk_W}}(q_x) \to a_x$:
  1. Compute $\hat{y}_i = \hat{f}(\hat{z}_i) = \mathsf{EvalApprox}_s(\mathsf{pk}, f', \hat{z}_i, \hat{\mathbf{r}}_{\text{aux}})$ for $i \in [2t]$.
  2. $a_x = (\hat{y}_1, \ldots, \hat{y}_{2t})$.
- $\mathsf{VC.Verify}_{\mathsf{sk_D}}(s_x, a_x)$:
  1. Check if $\hat{w}_i = \hat{y}_i$ for all $i \in [t]$.
  2. Check if $\mathsf{HE}'.\mathsf{Dec}_{\mathsf{sk}}(\hat{y}_{\pi(t+1)}) = \cdots = \mathsf{HE}'.\mathsf{Dec}_{\mathsf{sk}}(\hat{y}_{\pi(2t)})$.
  3. If either of the two tests above fails, return $\perp$; otherwise return $\mathsf{HE}'.\mathsf{Dec}_{\mathsf{sk}}(\hat{y}_{\pi(t+1)})$.

Fig. 2. One-Time Delegation Scheme

Lemma 4.3 (Completeness of $\mathcal{VC}$). *The verifiable computation scheme $\mathcal{VC}$ in Figure 2 has overwhelming completeness (Definition 2.8) for the class* $\mathsf{AC}^0_Q[2]$.

Proof. The proof is straightforward and stems directly from the homomorphic properties of HE' (Theorem 3.4). In fact, by construction and by definition of HE' (Section 3.4), the distribution of the $\hat{w}_i$-s is identical to $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(F, \hat{r}_i)$. Analogously, the distribution of $\hat{y}_i$-s is identical to $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(F, \hat{z}_i)$. □

Remark 4.3 (Efficiency of $\mathcal{VC}$). In the following we consider the verifiable computation of a function $F : \{0,1\}^n \to \{0,1\}^m$ computable by an $\mathsf{AC}^0_\mathsf{Q}[2]$ circuit of size S.

- $\mathsf{VC.KeyGen}$ can be computed by an NC^1 circuit of size $O(\mathsf{poly}(\lambda)S)$;
- $\mathsf{VC.ProbGen}$ can be computed by an $\mathsf{AC}^0[2]$ circuit of size $O(\mathsf{poly}(\lambda)(m+n))$;
- $\mathsf{VC.Compute}$ can be computed by an NC^1 circuit of size $O(\mathsf{poly}(\lambda)S)$;
- $\mathsf{VC.Verify}$ can be computed by a TC^0 circuit of size $O(\mathsf{poly}(\lambda)(m+n))$ and whose (constant) depth is independent of the depth of F.

Lemma 4.4 (One-time Soundness). *Under the assumption that* $\oplus\mathsf{L/poly} \subsetneq \mathsf{NC}^1$ *the scheme in Figure 2 is* $(1,1)$*-sound (one time secure) against* NC^1 *adversaries whenever t is chosen to be* $\omega(\log(\lambda))$.

Proof. We follow the same proof structure as in the proof of Lemma 12 in [CKV10]. We will keep part of the analysis informal, emphasizing why this proof still works for low-depth circuits. We refer the reader to [CKV10] for further details.

The following observation will be crucial in the rest of the proof. Notice that, by construction and by definition of HE' (Section 3.4), the distribution of the $\hat{w}_i$-s is identical to $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(F, \hat{r}_i)$. Analogously, the distribution of $\hat{y}_i$-s is identical to $\mathsf{HE}'.\mathsf{Eval}_{\mathsf{pk}}(F, \hat{z}_i)$.

Consider an NC^1 adversary $\mathcal{A}^*$ that cheats with non-negligible probability in the one-time security experiment $\mathbf{Exp}^{\mathsf{Verif}}_A[\mathcal{VC}, f, \lambda, 1, 1]$ (Definition 2.9). Let $(\hat{r}_1, \ldots, \hat{r}_t)$ be the independent copies of $\mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(\bar{0})$ and $(\hat{r}_{t+1}, \ldots, \hat{r}_{2t})$ the t independent copies of $\mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(x)$ as above. Whenever the verification algorithm accepts, the adversary must have responded correctly on $\hat{r}_1, ..., \hat{r}_t$ and incorrectly (and consistently) on $\hat{r}_{t+1}, \ldots, \hat{r}_{2t}$. Our goal is to bound the probability that the adversary succeeds in doing that.

First, notice that the view of the adversary is $(\mathsf{pk}_\mathsf{W}, \hat{r}_1, \ldots, \hat{r}_{2t})$, and identical to $(\mathsf{pk}_\mathsf{W}, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(\bar{0})^t, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(x)^t)$. By semantic security of the homomorphic encryption scheme, there exists an infinitely large set of parameters Λ such that $(\mathsf{pk}_\mathsf{W}, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(\bar{0})^t, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(x)^t) \sim_\Lambda (\mathsf{pk}_\mathsf{W}, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(\bar{0})^{2t})$. Consider a modified game where the adversary receives $(\mathsf{pk}_\mathsf{W}, \mathsf{HE}'.\mathsf{Enc}_{\mathsf{pk}_\mathsf{W}}(\bar{0})^{2t})$. Denote by p the probability that the adversary succeeds in this game. By computational indistinguishability we have

$$\Pr[\mathcal{A}^* \text{ is correct on } (\hat{r}_1, \ldots, \hat{r}_t) \text{ and incorrect on } (\hat{r}_{t+1}, \ldots, \hat{r}_{2t})] \leq p + \mathsf{neg}(\lambda)$$

for all $\lambda \in \Lambda$. This inequality holds because we can test in NC^1 whether $\mathcal{A}^*$ cheats only on $(\hat{r}_{t+1}, \ldots, \hat{r}_{2t})$. Therefore, if the adversary's behavior differed significantly between the two games, one would be able to break the semantic security of the homomorphic scheme. Here we made use of the third fact in Lemma 2.1.

We now proceed to upper bound p. Observe that

$$p = \Pr[\mathcal{A}^* \text{ is correct on } (\hat{z}_{\pi(1)}, \ldots, \hat{z}_{\pi(t)}) \text{ and incorrect on } (\hat{z}_{\pi(t+1)}, \ldots, \hat{z}_{\pi(2t)})]$$

where the $\hat{z}_{\pi(i)}$-s are defined as in Figure 2. Because of Lemma 4.1 that the distribution of π is statistically indistinguishable from that of a uniformly random permutation. Also, observe that the answers $\hat{y}_i$ of the adversary are independent of π. We can then conclude that $p \leq \frac{1}{\binom{2t}{t}} + \mathsf{neg}(t)$, which concludes the security analysis. □

4.2 A Reusable Verification Scheme

We now describe how to obtain a reusable verification scheme $\overline{\mathcal{VC}}$ applying the transformation in [CKV10] from one-time sound verification schemes through fully homomorphic encryption. The core idea behind the transformation in [CKV10] is to encapsulate all the operations of a one-time verifiable computation scheme through homomorphic encryption. We instantiate this transformation with the one-time verifiable construction $\mathcal{VC}$, described in Figure 2, and the simplest of our two somewhat homomorphic encryption schemes, HE (defined in Section 3.1).

Let $\mathcal{VC}$ be the verifiable computation scheme defined in Figure 2. The reusable verifiable computation scheme $\overline{\mathcal{VC}}$ = ($\overline{\mathsf{VC.KeyGen}}$, $\overline{\mathsf{VC.ProbGen}}$, $\overline{\mathsf{VC.Compute}}$, $\overline{\mathsf{VC.Verify}}$) is defined as follows.

- $\overline{\mathsf{VC.KeyGen}}(1^\lambda, f) \to (\mathsf{pk_W}, \mathsf{sk_D})$: The key generation stage is the same as in $\mathcal{VC}$.
- $\overline{\mathsf{VC.ProbGen}}_{\mathsf{sk_D}}(x) \to (\overline{q_x}, \overline{s_x})$:
  1. $(q_x, s_x) \leftarrow \mathsf{VC.ProbGen}_{\mathsf{sk_D}}(x)$;
  2. Compute a fresh pair of keys $(\mathsf{pk}_x, \mathsf{sk}_x) \leftarrow \mathsf{HE.Keygen}(1^\lambda)$;
  3. Compute $\hat{q}_x \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}_x}(q_x)$;
  4. $\overline{q_x} \leftarrow (\mathsf{pk}_x, \hat{q}_x)$; $\overline{s_x} \leftarrow (s_x, \mathsf{sk}_x)$
- $\overline{\mathsf{VC.Compute}}_{\mathsf{pk_W}}(\overline{q_x}) \to \overline{a_x}$:
  1. $\hat{a}_x \leftarrow \mathsf{HE.Eval}_{\mathsf{pk}_x}(\mathsf{VC.Compute}(\cdot, f), \hat{q}_x)$.
  2. $\overline{a_x} \leftarrow \hat{a}_x$.
- $\overline{\mathsf{VC.Verify}}_{\mathsf{sk_D}}(\overline{s_x}, \overline{a_x})$:
  1. $a_x \leftarrow \mathsf{HE.Dec}_{\mathsf{sk}_x}(\hat{a}_x)$.
  2. return $\mathsf{VC.Verify}_{\mathsf{sk_D}}(s_x, a_x)$.

Fig. 3. Transformation from one-time $\mathcal{VC}$ scheme to a *reusable* $\mathcal{VC}$ scheme

Corollary 4.1 (Completeness of $\overline{\mathcal{VC}}$). *The verifiable computation scheme $\overline{\mathcal{VC}}$ in Figure 3 has overwhelming completeness (Definition 2.8) for the class $\mathsf{AC}^0_Q[2]$.*

Proof. The completeness of the scheme above follows directly from the completeness of $\mathcal{VC}$ and the homomorphic properties of HE. Notice that we can use $\mathsf{HE.Eval}$ to homomorphically compute $\mathsf{VC.Compute}$ as the latter carries out a computation in $\mathsf{AC}^0_{\mathrm{CM}}[2]$ (although it is *approximating* a computation in $\mathsf{AC}^0_{\mathrm{Q}}[2]$). □

Remark 4.4 (Efficiency of $\overline{\mathcal{VC}}$). The efficiency of $\overline{\mathcal{VC}}$ is analogous to that of $\mathcal{VC}$ with the exception of a circuit size overhead of a factor $O(\lambda)$ on the problem generation and verification algorithms and of $O(\lambda^2)$ for the computation algorithm. All algorithms in $\overline{\mathcal{VC}}$ are computable by constant depth circuit (of unbounded fan-in) and the depth of the verification algorithm is independent of the function F.

Theorem 4.1. *Under the assumption that $\oplus\mathsf{L/poly} \subsetneq \mathsf{NC}^1$ the scheme $\overline{\mathcal{VC}}$ in Figure 3 is $(O(1), \mathsf{poly}(\lambda))$-sound (many-times secure) against NC^1 adversaries whenever t is chosen to be $\omega(\log(\lambda))$ in the underlying scheme $\mathcal{VC}$.*

Proof. By Lemma 4.4 there exists an infinite set $\Lambda \subseteq \mathbb{N}$ of security parameters for which $\mathcal{VC}$ "is secure". By the proof of Lemma 4.4, this set is also the set of parameters where the somewhat homomorphic encryption scheme HE "is secure". We will show that for all values in this same set Λ, the probability of success of any NC^1 adversary in $\mathbf{Exp}_A^{\mathsf{Verif}}[\overline{\mathcal{VC}}, f, \lambda, O(1), \mathsf{poly}(\lambda)]$ is negligible.

Assume by contradiction there exists an NC^1 adversary $\mathcal{A}^*$ that achieves non-negligible advantage in $\mathbf{Exp}_A^{\mathsf{Verif}}[\overline{\mathcal{VC}}, f, \lambda, O(1), \mathsf{poly}(\lambda)]$ for some $\lambda \in \Lambda$.

Claim: If $\overline{\mathcal{VC}}$ is not secure for some $\lambda^* \in \Lambda$ then we can break the one-time security of $\mathcal{VC}$. Let $l = O(1)$ be the number of rounds in the many-time soundness experiment for $\overline{\mathcal{VC}}$. Consider the following NC^1 adversary $\mathcal{A}_1$ for the experiment $\mathbf{Exp}_A^{\mathsf{Verif}}[\mathcal{VC}, f, \lambda, 1, 1]$:

- $\mathcal{A}_1$ obtains a pair a public key $\mathsf{pk_W}$ and sends it to $\mathcal{A}^*$;
- For all rounds $i \in \{1, \ldots, l-1\}$, $\mathcal{A}_1$ replies to $\mathcal{A}^*$ queries by generating a fresh pair of keys $(\mathsf{pk}, \mathsf{sk})$ and sending back encryptions of $\mathsf{HE.Enc_{pk}}(\bar{0})$;
- At round l, $\mathcal{A}_1$ responds to all input queries but the last one as above. This, by experiment definition, is the input where $\mathcal{A}^*$ will try to cheat; we denote this input by x^*. Now $\mathcal{A}_1$ sends x^* as the only input query in the one-time security experiment and will receive back q^*. It will then obtain a fresh pair of keys $(\mathsf{pk}^*, \mathsf{sk}^*)$ and send $\mathsf{HE.Enc_{pk^*}}(q^*)$ to $\mathcal{A}^*$.
- $\mathcal{A}^*$ will respond with $\hat{a}^*$ and $\mathcal{A}_1$ will send $\mathsf{HE.Dec_{sk^*}}(\hat{a})$ to the challenger for one-time security experiment.

The advantage of $\mathcal{A}_1$ depends on how likely is $\mathcal{A}^*$ can successfully cheat in that interaction. Let p be the advantage of $\mathcal{A}_1$ in the one-time security experiment. Clearly, if p is close to the advantage of $\mathcal{A}^*$ in the many-times security experiment $\mathcal{A}_1$ breaks the security of the one-time scheme.

Claim: the advantage of $\mathcal{A}_1$ is negligibly close to that of $\mathcal{A}^*$ in the many-time security game for security parameter λ^*. We can prove this by relying on the semantic security of the homomorphic encryption and on a hybrid argument.

Let $L = lm$, the total number of input queries in the many-times security experiment. We now define the hybrids $H^{(j)}$ with $j \in \{0, \ldots, L\}$. We define $H^{(0)}$ to be the exactly the many-time security experiment. For $j \in [L]$ we define $H^{(j)}$ to be an experiment where we respond to input queries with $\mathsf{HE.Enc}_{\mathsf{pk}_f}(\bar{0})$ where pk_f is a fresh public key up to input query j and behaves the many-time security experiment from input query $j+1$ on. Notice that $H^{(L)}$ corresponds to the interaction with $\mathcal{A}_1$ above.

Denote by $A^{(j)}$ the output distribution of $\mathcal{A}^*$ when interacting with $H^{(j)}$. Intuitively, if the advantage of the $\mathcal{A}_1$ in the one-time experiment is significantly different from the advantage of $\mathcal{A}^*$ in the many-times security games, then $A^{(0)}$ and $A^{(L)}$ are not Λ-computationally indistinguishable.

Therefore (by Lemma 2.1), there exists $j \in [L]$ such that $A^{(j-1)} \not\sim_\Lambda A^{(j)}$.

Claim: If there exists $j \in [L]$ such that $A^{(j-1)} \not\sim_\Lambda A^{(j)}$ then we can break the semantic security of HE. Consider the following NC^1 adversary $\mathcal{A}_{\text{CPA}}$ which receives in input a "challenge" public key pk^*. $\mathcal{A}_{\text{CPA}}$ will interact with $\mathcal{A}^*$ simulating $H^{(j)}$ until receiving input query x_j. At this point it will compute q_j from $\mathsf{VC.ProbGen}(x_j)$ and send to the CPA challenger (see Remark 2.1) q_j and $\bar{0}$, receiving back an encryption c^* of either message under the public key pk^*. $\mathcal{A}_{\text{CPA}}$ will now send (pk^*, c^*) to $\mathcal{A}^*$ and continue simulating $H^{(j)}$ till the end of the experiment. The adversary $\mathcal{A}_{\text{CPA}}$ will check whether $\mathcal{A}^*$ cheated successfully at the end of the experiment and output (in the multiple-message CPA experiment) 1 if that is the case and 0 otherwise. This would allow $\mathcal{A}_{\text{CPA}}$ to have a noticeable advantage in the experiment thus breaking the semantic security of HE. □

References

[ACK+02] David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *International Colloquium on Automata, Languages, and Programming*, pages 152–163. Springer, 2010.

[AM12] Pablo Daniel Azar and Silvio Micali. Rational proofs. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1017–1028. ACM, 2012.

[AM13] Pablo Daniel Azar and Silvio Micali. Super-efficient rational proofs. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 29–30. ACM, 2013.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In

Amit Sahai, editor, *Theory of Cryptography*, pages 315–333, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BP11] Joan Boyar and Rene C Peralta. A depth-16 circuit for the aes s-box. *IACR Cryptology ePrint Archive*, 2011(IACR Cryptology ePrint Archive), 2011.

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 24, page 39, 2017.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[CG15] Matteo Campanelli and Rosario Gennaro. Sequentially composable rational proofs. In *International Conference on Decision and Game Theory for Security*, pages 270–288. Springer, 2015.

[CG17] Matteo Campanelli and Rosario Gennaro. Efficient rational proofs for space bounded computations. In *International Conference on Decision and Game Theory for Security*, pages 53–73. Springer, 2017.

[CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, volume 6223, pages 483–501. Springer, 2010.

[CM97] Christian Cachin and Ueli Maurer. Unconditional security against memory-bounded adversaries. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 292–306, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112. ACM, 2012.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[DVV16] Akshay Degwekar, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Fine-grained cryptography. In *Annual Cryptology Conference*, pages 533–562. Springer, 2016.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford University, 2009.

[GGH+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N Rothblum. Verifying and decoding in constant depth. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 440–449. ACM, 2007.

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010*, pages 465–482. Springer, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 113–122. ACM, 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2001.

[Gol09] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge university press, 2009.

[GR18] Oded Goldreich and Guy N Rothblum. Simple doubly-efficient interactive proof systems for locally-characterizable sets. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 94. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[Hag91] Torben Hagerup. Fast parallel generation of random permutations. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 405–416, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[Has87] Johan Hastad. One-way permutations in nc0. *Information Processing Letters*, 26(3):153–155, 1987.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 294–304. IEEE, 2000.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Structure in Complexity Theory Conference, 1995., Proceedings of Tenth Annual IEEE*, pages 134–147. IEEE, 1995.

[LTKS15] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 706–719. ACM, 2015.

[Mer78] Ralph C Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[MV91] Yossi Matias and Uzi Vishkin. Converting high probability into nearly-constant time - with applications to parallel hashing. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 307–316, New York, NY, USA, 1991. ACM.

[Raz87] Alexander A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[Yao82] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.