From Secrecy to Soundness: Efficient Verification via Secure Computation (Extended Abstract)

Benny Applebaum[1⋆], Yuval Ishai[2⋆⋆], and Eyal Kushilevitz[3⋆⋆⋆]

[1] Computer Science Department, Weizmann Institute of Science
[2] Computer Science Department, Technion and UCLA
[3] Computer Science Department, Technion

Abstract. We study the problem of *verifiable computation* (VC) in which a computationally weak client wishes to delegate the computation of a function f on an input x to a computationally strong but untrusted server. We present new general approaches for constructing VC protocols, as well as solving the related problems of program checking and self-correcting. The new approaches reduce the task of verifiable computation to suitable variants of secure multiparty computation (MPC) protocols. In particular, we show how to efficiently convert the secrecy property of MPC protocols into soundness of a VC protocol via the use of a message authentication code (MAC). The new connections allow us to apply results from the area of MPC towards simplifying, unifying, and improving over previous results on VC and related problems.
In particular, we obtain the following concrete applications: (1) The first VC protocols for arithmetic computations which only make a black-box use of the underlying field or ring; (2) a non-interactive VC protocol for boolean circuits in the preprocessing model, conceptually simplifying and improving the online complexity of a recent protocol of Gennaro et al. (Cryptology ePrint Archive: Report 2009/547); (3) $\mathbf{NC^0}$ self-correctors for complete languages in the complexity class $\mathbf{NC^1}$ and various log-space classes, strengthening previous $\mathbf{AC^0}$ correctors of Goldwasser et al. (STOC 2008).

1 Introduction

In the *verifiable computation* (VC) problem, we have a computationally weak device (client) who wishes to compute a complex function f on an input x. The client is too weak to compute f on its own and so it delegates the computation to a computationally strong server. However, the client does not trust the server

⋆ Work done in part while visiting Princeton University. Supported by Koshland Fellowship, and NSF grants CNS-0627526, CCF-0426582 and CCF-0832797.
⋆⋆ Supported by BSF grant 2008411, ISF grant 1310/06, and NSF grants 0830803, 0716835, 0627781.
⋆⋆⋆ Work done in part while visiting UCLA. Supported by BSF grant 2008411 and ISF grant 1310/06.

and therefore would like to be able to verify the correctness of the computation without investing too much resources. One may also consider a stronger variant of the problem in which, in addition to the ability to detect arbitrary errors, the client should be able to *correct* the errors as long as the server is somewhat correct with respect to some predefined distribution over the inputs. This corresponds to the scenario where the server, or alternatively a program locally run by the client, makes "unintentional" errors on some fraction of the inputs (e.g., due to implementation bugs). Still, a malicious server should not be able to fool the client to accept an erroneous answer. We refer to this variant as the *correctable verifiable computation* (CVC) problem.

VC and CVC are fundamental problems which were extensively studied in various settings, originating from the early works on interactive proofs [4, 21] and program checking [7, 9, 28]. Recent advances in technology further motivate these problems. On the one hand, computationally weak peripheral devices such as smart phones and netbooks are becoming increasingly common; on the other hand, the increasing use of distributed computing over the internet also makes strong servers more commonly available. Indeed, the growing volume of outsourcable computation in the form of "cloud computing" or in projects like SETI@Home has attracted a renewed interest in the VC problem, and a considerable amount of research was devoted to these problems in the last few years [22, 18–20, 15, 11]. See [20, 15] for further discussion of applications as well as a survey of related work.

In this work, we present new general approaches for solving the VC and CVC problems, as well as the related problems of program checking and self-correcting. Our approaches employ variants of secure multi-party computation (MPC), converting their *secrecy* features into *soundness* by making additional use of basic cryptographic primitives such as message authentication codes (MACs) and symmetric encryption. By instantiating these general approaches we obtain several improvements over previous results in this area. We stress that the idea of employing secrecy in the context of verification is not unique to our work. This idea can be traced back to the first works on interactive proofs and program checking [4, 21, 7, 28] and is also implicit in more recent works in the area [18, 19, 15, 11]. Our work provides *new* approaches for converting secrecy into soundness that have advantages of generality, efficiency, and simplicity over previous approaches.

1.1 Background

Before introducing our new approaches to VC, we review some of the relevant notions and previous approaches.

MPC and related primitives. A protocol for secure two-party computation [32, 17] allows two parties, each holding a private input x_i, to compute a function on their joint input without revealing any additional information to each other. That is, the first (resp., second) party learns the output of some predefined function $f_1(x_1, x_2)$ (resp., $f_2(x_1, x_2)$) without learning any additional information about

x_2 (resp., x_1). Unless otherwise mentioned, we only require *computational* security against *semi-honest* parties who operate as instructed by the protocol (but may try to learn additional information from the messages they observe), and make no secrecy or correctness requirements in the presence of malicious parties.

We will be interested in secure protocols in which one of the parties is restricted in its computational resources in a way that prevents it from computing the output on its own, even when given the entire input. Such restrictions may include bounds on sequential or parallel time (either with or without preprocessing), on space complexity, on arithmetic circuit complexity, etc. We will refer to the weak party as the *client* and to the strong party as the *server*. In contrast to the typical study of feasibility questions in the area of secure computation, in the context of restricted clients it makes sense to consider even functions for which only the client holds an input, as well as protocols for such functions with perfect or statistical rather than computational security. (The existence of statistically secure two-party protocols can be ruled out for almost all natural functions which depend on both inputs.)

A client-server protocol in which only the client has an input and gets an output is called an *instance-hiding* (IH) protocol [1, 5]. For simplicity, we will mainly restrict the attention to *one-round* (or two-message) IH protocols which consist of a single "query" from the client to the server followed by a single "answer" from the server to the client. A natural extension to multi-round IH protocols is deferred to the full version.

Central to this work is a different (and in some sense more stringent) variant of client-server protocols, in which only the client has an input x but *both parties* learn the same output $f(x)$. One-round protocols of this type coincide with the notion of *randomized encoding* from [23, 3]. A randomized encoding (RE) of f is a function $\hat{f}(x; r)$ whose output on a uniformly random and secret r can be used to decode $f(x)$ but reveals no additional information about x. In the corresponding client-server protocol, the client picks r at random and sends the encoded output $\hat{f}(x; r)$ as a query to the server; the server decodes the output $f(x)$ and sends it back as an answer to the client. In the full version, we discuss applications of an interactive variant of this primitive, referred to as *interactive randomized encoding* (IRE).[4] RE and IRE protocols can be easily converted into IH protocols with the same number of rounds by modifying the function f to compute an *encryption* of the output under a secret key selected by the client. A similar transformation in the other direction seems unlikely. As a notable example, the existence of a *fully homomorphic encryption* scheme [16] implies a one-round IH protocol for any polynomial-time computable f in which the client's time complexity grows only linearly with the input length, whereas the existence of similar RE protocols is an open problem.

Note that for all of the above types of client-server protocols, we are not concerned with protecting the privacy of the server, since the server has no

[4] This generalization is somewhat subtle in that it involves a nontrivial security requirement against a malicious server; see full version for details.

input. We can therefore assume, without loss of generality, that an honest server is deterministic.

The traditional approach for VC. The literature on program checking and interactive proofs already makes an implicit use of a general transformation from IH to VC.[5] For simplicity, we restrict the attention to one-round IH protocols. The basic idea is roughly as follows. The client uses the IH protocol to compute $f(x)$ while hiding the input x from the server, except that it randomly mixes the "real" IH query with an appropriately-distributed random query whose correct answer is somehow known (more on this below). The client accepts the output obtained from the real IH instance only if the server's answer on the dummy query is identical to the precomputed answer. By the hiding property, the server cannot distinguish the real query from the dummy one, and so a cheating server will be caught with probability $\frac{1}{2}$. (The soundness can be amplified via repetition.) More formally, this approach requires two building blocks: (1) a one-round IH protocol, in which the client efficiently maps x to a query $\hat{x}$ such that, given the server's answer $g(\hat{x})$ (together with the client's randomness), it is possible to efficiently recover $f(x)$; and (2) a *solved instance generator* (SIG): an efficient way for generating a random instance r for g (under the distribution defined by the client's query in the IH scheme) together with its solution $g(r)$.

We summarize the advantages and disadvantages of the SIG+IH approach. On the positive side, IH is a relatively liberal notion which is implied by secure computation in the semi-honest model, and SIG is easy in many cases, e.g., it is given "for free" if polynomial-time preprocessing is allowed before *each* real query. (See [11] for a the usefulness of this approach when applied with IH based on fully homomorphic encryption, and [15, 11] for the further use of fully homomorphic encryption towards *reusable* preprocessing.) On the negative side, SIGs are not *always* easy to construct (for instance, the absence of parallel SIGs significantly complicated the parallel checkers of [19] and prevented [19] from achieving highly parallel correctors for, say, log-space complexity classes). Another disadvantage of the SIG+IH approach has to do with the overhead of soundness amplification: in order to achieve soundness error of $2^{-\tau}$, the VC protocol needs to invoke the IH and SIG protocols $\Omega(\tau)$ times.

1.2 Our solutions

Motivated by the above disadvantages of the traditional approach, we present two new approaches for transforming variants of MPC into VC or CVC.

Construction 1: VC from RE+MAC. Our first approach is based on a novel combination of an RE (or IRE protocol) with a private-key signature scheme (also known as message authentication code or MAC). Unlike previous approaches, we employ secrecy in order to hide the MAC's secret key, rather than the inputs

[5] The following formulation is similar to the one from Section 1.2 of [18]; see [9, 14, 19, 11] for other variants and applications of this approach.

of the computation. The idea is as follows: Given an input x, the client asks the server to compute $y = f(x)$ and, in addition, to generate a signature on $f(x)$ under a private key k which is chosen randomly by the client. The latter request is computed via an RE protocol that hides the private key from the server. More precisely, the client who holds both x and k, invokes an RE such that both parties learn the function $g(x, k) = \mathtt{MAC}_k(f(x))$. The client then accepts the answer y if and only if the result of the protocol is a valid signature on y under the key k. The soundness of the protocol follows by showing that a cheating server, which fools the client to accept an erroneous $y^* \neq f(x)$, can be used to either break the privacy of the RE or to forge a valid signature on a new message. For this argument to hold, it is crucial for the RE to be secure in the following sense: a malicious server should not be able to force an erroneous output which violates privacy; that is, one should be able to simulate erroneous outputs solely based on the correct outputs. In the case of RE (where there are only two messages), this requirement follows automatically from the basic secrecy requirement against a *semi-honest* server. In the interactive setting, we show that such useful IRE protocols can be extracted from various MPC protocols that appear in the literature.

Note that the above approach eliminates both of the disadvantages of the SIG+IH approach mentioned above, at the expense of replacing IH with the stronger RE primitive and (slightly) increasing the complexity of the function f by applying a MAC computation to its output.

Construction 2: CVC from RE + One-time pad. The previous construction does not seem to apply to the case of CVC. Our second construction yields a CVC protocol and, as can be expected, is somewhat less efficient. The starting point is the well-known CVC version of the SIG+IH approach [9]. In this version, dummy IH queries obtained via SIG are mixed with (randomized) IH queries for the real instance. The client first verifies that most of the dummy queries were answered correctly, and then outputs the majority vote of the outputs obtained from the real answers. Our main goal here is to eliminate the need for SIG primitive. The idea is to generate the dummy queries by applying the IH to some default input x_0 whose image $y_0 = f(x_0)$ is known, and compare the *outputs* obtained from these dummy queries to the known output y_0. (The fixed value of y_0 can be "wired" into the description of the client and used in all subsequent invocations.) By the hiding property the messages of the client are distributed according to some fixed universal probability distribution D which does not depend on the actual input. By using standard concentration bounds, one can show that the client will correct the errors of a "buggy" (rather than malicious) server which doesn't err too much over messages drawn from D. Intuitively, the privacy of the IH protocol also prevents a *malicious* server from cheating, as such a server cannot distinguish between a "dummy" query to a "real" one, and therefore a cheating behavior will be be detected (whp). However, this intuition is inaccurate as, in general, even if the server cannot distinguish dummy queries from real ones, it might be able to apply the same strategy to all the queries such that errors will

be generated only in the real queries.[6] Fortunately, this can be fixed by requiring an additional *sensitivity* property: any erroneous message of the server should lead to an erroneous answer of the client. To achieve this property, we combine an RE protocol with a one-time pad encryption scheme. That is, we employ an RE for the function $g(x,k) = k \oplus f(x)$ where k is used as a one-time pad. The use of one-time pad transforms the RE to a "sensitive" IH.

Compared to the traditional approach, the above approach eliminates the need for SIG at the expense of strengthening the IH primitive.

2 Applications

By instantiating our generic approaches, we derive new constructions of VC and CVC protocols in several settings.

2.1 Online/offline non-interactive VC

In the online/offline setting [20, 15], the client can afford to invest a lot of computational resources in a preprocessing phase before seeing the actual input x, but only a small amount of computational resources after x is known. (Imagine a smart card which is initialized in a secure environment and later operates in a hostile environment with an untrusted server.) Since communication may also be limited, especially for weak devices, we would like the protocol to be non-interactive. That is, in the offline phase the client should perform some (possibly expensive) computation and send the result (the "public-key") to the server or publish it somewhere.[7] In the online phase, when the client obtains its input x, it should send a single computationally-cheap message to the server. The server then computes the result without any intermediate interaction with the client, which in the meantime can be disconnected from the network. At the end of the computation, the server publishes an answer. Based on this answer, the client recovers the result $y = f(x)$ or announces an error in the case of a cheating server.

We would like to minimize the client's online time complexity ideally to be only linear in the input and output length of f. We also require the complexity of the server to be polynomial in the time complexity of f. There are only few known solutions that yield almost optimal non-interactive VCs (NIVCs) for general Boolean functions. These include the constructions of Micali [30]

[6] Consider, for example, an IH in which a client whose input equals to the all zero string, ignores the server's answers and outputs $f(\mathbf{0})$. A CVC protocol which makes use of such an IH together with $x_0 = \mathbf{0}$ can be trivially broken by a malicious server which sends erroneous answers.

[7] In a concurrent and independent work, Chung, Kalai, and Vadhan [11] obtain a qualitatively stronger type of non-interactive VC protocols, where the offline preprocessing phase can only involve a *local* computation performed by the client with no additional interaction. The applications we present only apply to the weaker model of non-interactive VC, but obtain better online efficiency in this model.

in the random oracle model, the construction of Goldwasser et al. and Kalai and Raz [20, 26] for low-depth circuits, and the recent construction by Gennaro et al. [15] for polynomial-size Boolean circuits which relies on the existence of one-way functions.

While these constructions provide good solutions for binary computations, they suffer from large overhead in the case of arithmetic computations. Indeed, a client who wishes to delegate a computational task which should be performed over non-binary domains such as the integers, finite-precision reals, matrices, or elements of a big finite ring, has no choice but to translate the computation into a binary circuit and then apply one of the above solutions. This results in large computational and communication overhead which heavily depends on the exact structure of the underlying ring.[8] A much more satisfactory solution would be to describe the computation in an arithmetic model in which computational operations are performed over some ring $\mathcal{R}$ and then employ an arithmetic NIVC. More formally, we would like to have a protocol in which both the server and the client only have a *black-box* access to $\mathcal{R}$. This black-box access enables the client and server to perform ring operations and sample random ring elements, but the correspondence between ring elements and their identifiers (or even the exact size of the ring) will be unknown to the algorithms. The black-box ring model allows to abstract away the exact structure of the underlying ring, and thus to obtain protocols in which the number of ring operations does not depend on the actual algebraic structure of $\mathcal{R}$. Unfortunately, all the above constructions do not seem to achieve such a result. The reason is that the main tools employed by these constructions (i.e., PCP machinery in the case of [30, 20], and Yao's garbled circuit [32] in the case of [15]) do not seem to work in the arithmetic black-box model, even for the special case of black-box fields.

Our results. We obtain NIVCs in the black-box ring model for arithmetic branching programs [6] (ABPs) which are the arithmetic analog of log-space counting classes.[9]

Theorem 1 (informal). *Assuming the existence of one-way functions, there exists a NIVC in the BBR model with perfect completeness and computational soundness error* $\mathrm{neg}(\tau)$ *where* τ *is the security parameter. The complexity of the offline phase and the server's complexity are* $\mathrm{poly}(s, \tau)$*, the time complexity of the online phase is* $O(n\tau)$ *at the query step and* $O(\tau)$ *at the verification step, where* n *is the input length, and* s *is the size of the ABP.*

To the best of our knowledge, this is the first construction of VC in the black-box arithmetic model, even for the case of black-vox fields and even if many rounds of interaction are allowed. The main ingredient is a new construction of arithmetic REs with low online complexity (which is based on [24, 12]). The

[8] For example, even in the case of finite fields with n-bit elements, the size of the best known Boolean multiplication circuits is $\omega(n \log n)$; the situation is significantly worse for other useful rings, such as matrix rings.

[9] Such programs are quite expressive and are capable of emulating arithmetic formulas.

NIVC is obtained by plugging this RE (together with black-box arithmetic MAC) into our RE+MAC approach.

Optimized and simplified NIVC for Boolean circuits. As an additional application, we simplify the recent online/offline NIVC of Gennaro, Gentry and Parno [15] as well as improve its online efficiency. Specifically, GGP constructed a NIVC for every polynomial-size circuit $f : \{0,1\}^n \to \{0,1\}^m$ with low online complexity as well as low amortized offline complexity. This is achieved in two steps. First, a basic NIVC with low online complexity is constructed by relying on special properties of Yao's garbled circuit (GC) construction and, then, a fully homomorphic encryption scheme is used to reduce the amortized complexity of the offline phase. Our version of the basic protocol follows immediately by instantiating the RE+MAC approach with computationally-sound RE based on GC [2]. This leads to the following theorem:

Theorem 2 (informal). *Assuming the existence of one-way functions, every function $f : \{0,1\}^n \to \{0,1\}^m$ of circuit size s, can be realized by a NIVC with perfect completeness, computational soundness error of $\mathrm{neg}(\tau) + 2^{-\sigma}$ (where τ and σ are computational and statistical security parameters, respectively), and complexity as follows.* ***Client:*** *Offline complexity $O(s \cdot \tau + \sigma)$, and online complexity of $O(n \cdot \tau)$ at the query step, and $O(m + \sigma)$ at the verification step.* ***Server:*** *complexity of $O(s \cdot \tau + \sigma)$.*

This theorem simplifies and slightly improves the efficiency of the basic GGP scheme. Simplification comes from the fact that we do not need to rely on any specific properties of Yao's encoding other than its standard security and the well known ability to break the computation of the GC into an offline phase and a cheap online phase. Moreover, we also get an efficiency advantage: in the online phase of the GGP protocol the client needs to get an encryption key for each bit of the output. Hence, both the communication and computation complexity at the verification stage are $O(m\tau)$ where τ is a computational security parameter. In our case, the client needs to get (in addition to the output) only a short certification string of size σ, where σ is a *statistical* security parameter, and so the complexity is $O(m + \sigma)$. This difference can be significant for computations in which the output is much longer than the input (and shorter than the circuit size). For instance, think of image processing algorithms which return "enhanced" versions of low-quality pictures or movies.[10] Finally, we observe that the second step of the [15] construction in which the offline complexity is being amortized forms a general transformation and so it can be used to amortize the offline stage of our construction as well. (A similar observation was made independently and concurrently by [11].)

[10] One thing to note, though, is that if the client already knows a candidate y for $f(x)$ (obtained either from the server or from some other source) then the GGP approach can be applied for the boolean function $g(x, y)$ which verifies that $y = f(x)$. In such a case, the communication to the client will only be $m + O(\tau)$, but the online communication to the server grows asymptotically when y is longer than x.

Program checking and correcting. In the setting of program checking [7, 9], one would like to have a VC protocol for a function f in which the power of the honest server is limited: it can only compute the function f itself. That is, the honest server always responds to a message q by the message $f(q)$. Such a VC protocol is called *program self-checker*.[11] Indeed, a checker can be used to check the correctness of a possibly faulty program for f on a given input, by letting the program play the role of the server. Similarly, a CVC in which the server can be implemented by f is called a self-tester/corrector pair, as it allows to test whether a given program is not too faulty, and if so to correct it.

Minimizing the parallel complexity. Rubinfeld [31] initiated the study of the parallel complexity (circuit depth) of program checkers and correctors, and showed that some non-trivial functions can be checked by $\mathbf{AC^0}$ checkers (i.e., constant depth circuits with AND and OR gates of unbounded fan-in). Goldwasser et al. [19] proved several surprising results about the parallel complexity of program checking and correcting. Among other things, they showed that a rich family of combinatorial and algebraic languages, namely, all the complete languages in the complexity classes $\mathbf{NC^1}, \oplus\mathbf{L}/poly, \mathbf{Mod_kL}/poly$, can be checked in $\mathbf{NC^0}$ (i.e., by constant depth circuits with bounded-fan in gates) and corrected in $\mathbf{AC^0}$.[12] We improve this result by showing that all these languages can be also *corrected* in $\mathbf{NC^0}$:

Theorem 3 (informal). *Every language which is complete for one of the complexity classes $\mathbf{NC^1}, \oplus\mathbf{L}/poly, \mathbf{Mod_kL}/poly$ under $\mathbf{NC^0}$ Karp reductions can be checked, tested and corrected by an $\mathbf{NC^0}$ client with perfect completeness and (arbitrarily small) constant statistical soundness error. Correction succeeds with arbitrary large constant probability (say $2/3$) as long as the server's error probability is bounded away from $1/2$ (e.g., $1/3$).*

Furthermore, our corrector (and checker) only makes a constant number of calls to the program in a *non-adaptive* way. This is contrasted with the constructions of [19] which make an adaptive use of the program even in the case of checkers. (This difference seems to be inherent to the "composition approach" of [19] which breaks the computation to subroutines and checks them by subcheckers.) As a concrete example of our improvement, consider the function Det which computes the determinant of an $n \times n$ matrix over a field $\mathbb{F}_p$ of fixed prime order. Since Det is complete for the class $\mathbf{Mod_pL}/poly$ [29], we can get an $\mathbf{NC^0}$ tester/correcter for the determinant over any fixed finite field which makes a constant number of calls to the program. Previous correctors either had polynomial depth [9], or were implemented in $\mathbf{AC^0}$ and made large (non-constant) number of calls to the program [19]. (See [19, Table 1]). Our constructions are obtained by instantiating the RE+OTP approach with the $\mathbf{NC^0}$ REs of [3].

[11] In fact, the notion defined here is slightly stronger than the original definition of [7], and corresponds to *adaptive checkers* as in [8].

[12] Recall that there is a considerable gap between these two classes, as in $\mathbf{NC^0}$ circuits each bit of the output depends only on a constant number of input bits; thus, an $\mathbf{NC^0}$ circuit cannot compute even an n-bit AND gate.

Additional properties. We mention that most of our protocols satisfy additional useful properties. For example, we can add input-privacy and allow the client (or checker) employ the program without revealing its input. In some cases, we can also add a form of zero-knowledge property: the client learns only the value $f(x)$ and no other additional information that she cannot compute by herself using her own *weak* resources. This may be useful when the server is getting paid for his work and does not want to be abused and supply additional computation services for free during the VC protocol. These extensions are deferred to the full version.

3 Verifiable computation from RE and MAC

3.1 Definitions

Message Authentication Codes. A *one-time message authentication code* (MAC) is an efficiently computable function $\mathtt{MAC} : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ which maps a message $x \in \mathcal{M}$ and a random secret key $k \in \mathcal{K}$ to a signature $\sigma = \mathtt{MAC}_k(x) \in \mathcal{C}$. A MAC is statistically secure with error ε if for every $x \in \mathcal{M}$ and every computationally unbounded adversary A, $\Pr_k[A(x, \mathtt{MAC}_k(x)) = (y, \mathtt{MAC}_k(y)) \wedge (y \neq x)] \leq \varepsilon$.

Verifiable Computation. A *verifiable computation protocol* (VC) for a function f with soundness error $0 \leq \varepsilon \leq 1$ is an interactive protocol between a client C and a server P such that (1) perfect completeness[13]: for every input x the client outputs $f(x)$ at the interaction $(C, P)(x)$ with probability 1; and (2) soundness: for every input x of the client and every cheating P^*, we have $\Pr[(C, P^*)(x) \notin \{f(x), \perp\}] \leq \varepsilon$, where $\perp$ is a special "rejection" symbol and the probability is taken over the coin tosses of the client C. If P^* is restricted to be a polynomial-size circuit then the protocol is *computationally sound*.

Randomized Encoding [23, 3]. A *randomized encoding* (RE) for a function f is a non-interactive protocol in which the client uses its randomness r and its input x to compute a message $\hat{y} = \hat{f}(x; r)$ and sends it to the server, who responds by applying a decoder algorithm B to $\hat{y}$, recovers $f(x)$ and sends it back to the client. The protocol should satisfy: (1) *perfect completeness* (as in VC); and (2) ε*-privacy*: There exists a simulator S^* such that for every x the distribution $S^*(1^{|x|}, f(x))$ is at most ε-far (in statistical distance) from the distribution of the client's message $\hat{f}(x; r)$. *Computational privacy* is defined by restricting S^* to be a polynomial-size circuit and replacing statistical distance with ε-computational indistinguishability.

3.2 Our reduction

Our protocol is described in Figure 1.

[13] Due to space limitations, we always assume that protocols have perfect completeness. Our results hold in the more general setting where protocols have some completeness error δ.

- Primitives: MAC $\mathtt{MAC}$, and RE $\hat{g}$ for $g(k,x) = \mathtt{MAC}_k(f(x))$.
- Client's input: $x \in \{0,1\}^n$.

1. **Client**: Client chooses a random key k for $\mathtt{MAC}$, and random coins r and sends x together with the encoding $\hat{g}((k,x);r)$.
2. **Server**: Applies the decoder of $\hat{g}((k,x);r)$ and sends the result z together with $y = f(x)$.
3. **Client**: Accepts y if $\mathtt{MAC}_k(y)$ equals to z.

Fig. 1. A verifiable computation protocol for f based on a RE for $g(k,x) = \mathtt{MAC}_k(f(x))$.

The following lemma (whose proof is deferred to the full version) holds both in the statistical and computational setting:

Lemma 1. *Suppose that the RE and MAC have privacy errors of ε and ε', respectively. Then, the above protocol is a VC for f with soundness error $\varepsilon + \varepsilon'$.*

Proof (sketch). Fix a cheating server P^* and an input x^*. Let α be the probability that the client accepts some $y \neq f(x^*)$. We show that $\alpha \leq \varepsilon + \varepsilon'$. Consider the following attack on the MAC. Given x^* we compute $f(x^*)$ and ask for a signature $\mathtt{MAC}_k(f(x^*))$, where k is an unknown uniformly chosen MAC key. Then, we will use the RE simulator to simulate the encoding of $\mathtt{MAC}_k(f(x))$ up to distance ε and send the result together with x to P^*. Finally, output the pair (y,z) generated by P^*. Since the view of the adversary is ε-close to the view of P^* in a real interaction, the attack succeeds with probability at least $\alpha - \varepsilon$, which by the security of the MAC should be at most ε'. It follows that $\alpha \leq \varepsilon + \varepsilon'$.

Acknowledgements. We thank Guy Rothblum for useful discussions, and Yael Tauman Kalai for sharing with us a copy of [11].

References

1. M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. In *STOC*, 1987.
2. B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. *Computional Complexity*, 15(2):115–162, 2006.
3. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. *SICOMP*, 36(4):845–888, 2006.
4. L. Babai. Trading group theory for randomness. In *STOC*, 1985.
5. D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *STACS*, 1990.
6. A. Beimel and A. Gál. On arithmetic branching programs. *JCSS*, 59(2):195–220, 1999.
7. M. Blum and S. Kannan. Programs that check their work. In *STOC*, 1989.
8. M. Blum, M. Luby, and R. Rubinfeld. Program result checking against adaptive programs and in cryptographic settings. In *Distributed Computing and Crypthography: DIMACS Workshop*, 1990.

9. M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting programs with applications to numerical problems. In *STOC*, 1990.
10. G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel. Structure and importance of logspace-MOD-classes. In *STACS*, 1991.
11. K. M Chung, Y. T Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In submission, 2010.
12. R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In *EUROCRYPT*, 2003.
13. I. Damgård and J. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Crypto*, 2003.
14. J. Feigenbaum. Locally random reductions in interactive complexity theory. In *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages pp. 73–98, 1993.
15. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. Cryptology ePrint Archive, Report 2009/547, 2009. http://eprint.iacr.org/.
16. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.
17. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, 1987. 7.
18. S. Goldwasser, D. Gutfreund, A. Healy, T. Kaufman, and G. N. Rothblum. Verifying and decoding in constant depth. In *STOC*, 2007.
19. S. Goldwasser, D. Gutfreund, A. Healy, T. Kaufman, and G. N. Rothblum. A (de)constructive approach to program checking. In *STOC*, 2008.
20. S. Goldwasser, Y. T Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, 2008.
21. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SICOMP*, 18, 1989.
22. S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *TCC*, 2005.
23. Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, 2000.
24. Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, 2002.
25. Y. Ishai, M. Prabhakaran, and A. Sahai. Secure arithmetic computation with no honest majority. In *TCC*, 2009.
26. Y. T Kalai and R. Raz. Probabilistically checkable arguments. In *CRYPTO*, 2009.
27. M. Karchmer and A. Wigderson. On span programs. In *Structure in Complexity Theory Conference*, 1993.
28. R. J. Lipton. New directions in testing. In *Distributed Computing and Cryptography*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191–202. 1991.
29. M. Mahajan and V. Vinay. Determinant: combinatorics, algorithms and complexity. *Chicago J. Theoret. Comput. Sci.*, (5), 1997.
30. S. Micali. CS proofs (extended abstracts). In *FOCS*, 1994.
31. R. Rubinfeld. Designing checkers for programs that run in parallel. *Algorithmica*, 15(4):287–301, 1996.
32. A. C. Yao. How to generate and exchange secrets. In *FOCS*, 1986.