Verifiable Computation over Encrypted Data in the Presence of Verification Queries

Rosario Gennaro and Valerio Pastro

Center for Algorithms and Interactive Scientific Software, The City College of New York,
{rosario,pastro}@cs.ccny.cuny.edu

Abstract. We consider the problem of a client who outsources the computation of a function f over an input x to a server, who returns $y = f(x)$. The client wants to be assured of the correctness of the computation and wants to preserve confidentiality of the input x and possibly of the function f as well. Moreover, the client wants to invest substantially less effort in verifying the correctness of the result than it would require to compute f from scratch.
This is the problem of secure outsourced computation over encrypted data. Most of the work on outsourced computation in the literature focuses on either privacy of the data, using *Fully Homomorphic Encryption (FHE)*, or the integrity of the computation. No general security definition for protocols achieving both privacy and integrity appears in the literature. Previous definitions only deal with a very limited security model where the server is not allowed to issue *verification queries* to the client: i.e. it is not allowed to "see" if the client accepts or rejects the value y.
In this paper we present:

1. A formal definition of *private and secure* outsourced computation *in the presence of verification queries*;
2. A protocol based on FHE that achieves the above definition for arbitrary poly-time computations;
3. Some additional protocols for the computation of *ad-hoc* functions (such as the computation of polynomials and linear combinations) over encrypted data. These protocols do not use the power of FHE, and therefore are much more efficient than the generic approach. We point out that some existing protocols in the literature for these tasks become insecure in the presence of verification queries, while our protocols can be proven in the stronger security model where verification queries are allowed.

1 Introduction

Is it possible to delegate the *processing* of your data to a party you don't completely trust? What if you do not want to give full access to your data to such party? Or what if the computation you delegate is so sensitive that you must make sure the result is correct, but must do so using only very limited computational resources (which is the reason you delegated the computation in the first place)? These are central questions related to the security of *cloud computing*, a paradigm where businesses buy computing time from a service, rather than purchase and maintain their own computing resources. These issues also arise from the proliferation of mobile devices, such as smart phones and tablets: computationally weak devices which might outsource computationally intensive operation, e.g. a cryptographic operation or a photo manipulation that they are not able to perform on their own.

This might explain why *Secure Outsourced Computation* has become a very active research area in cryptography. When it comes to the security and privacy concerns associated with outsourced computation, we can summarize them in the following very important two questions:

- *Is it possible to protect the privacy of the input to the computation?* In other words, can an outside party compute for us, without learning our private data?
- *Can you efficiently verify the result of the computation?* In other words, how does one check that the outside party performed the computation correctly without investing too many computational resources in the verification (i.e. by redoing the computation from scratch)?

Both questions have a long research history associated with them. The need for an encryption algorithm that would allow arbitrary computation over encrypted data was recognized very early by cryptographers

(see the notion of *privacy homomorphism* in [RAD78]). A tantalizing research question, it remained open until the breakthrough result by Gentry [Gen09], who constructed the first *Fully Homomorphic Encryption (FHE)*. While the initial proposal remains only of theoretical interest, Gentry's work revealed a new set of techniques which were immediately put to use in the construction of many other more efficient schemes [Gen10,SV10,GH11,BV11,BGV12,GHS12,Bra12], and today we seem to stand at the verge of having schemes which can be used in practice.

Efficient verification of arbitrarily complex computations was the underlying goal of *Interactive Proofs* [Bab85,GMR89], where a powerful (e.g. super-polynomial) prover can (probabilistically) convince a weak (e.g. polynomial) verifier of the truth of statements that the verifier could not compute on its own. The goal to make the verifier as efficient as possible led to the concept of *probabilistically checkable proofs* (PCPs) [AS92,BFL90,BFLS91], where a prover can prepare a proof that the verifier needs to check in only very few places (in particular only a constant number of bits of the proofs needed for NP languages) and their cryptographic applications: the efficient arguments of Kilian [Kil92,Kil95] and Micali's non-interactive CS Proofs [Mic94].

The application to cloud computing has rekindled attention to this area, particularly in the search for protocols where arbitrary *poly-time* (as opposed to superpoly-time) computations can be efficiently verified by a linear (or quasi-linear) verifier, and performed by a prover without too much overhead. Starting with the work on *muggles proof* [GKR08] a line of research revisited and "scaled down" the PCP machinery [GLR11,BCCT12a,BCCT12b]. Another line of work explored alternative ways of arithmetizing computations to construct efficient proofs [Gro10,Lip12,GGPR13]. Yet another approach used FHE as a tool to build efficient verification of arbitrary computations [GGP10,AIK10,CKV10].

Several implementation efforts [CMT12,Tha13,PHGR13,BSCG+13] show that here too we are on the verge of achieving practical efficiency, with the *Quadratic Span Program* techniques of [GGPR13,PHGR13] showing particular promise.

1.1 Our Contribution

Given the practical and theoretical relevance of this research area, and the level of maturity the field is reaching, it is somewhat surprising to notice that all the papers mentioned above deal with either one of the two main questions. There are many results about finding efficient FHE schemes, and therefore efficient computation over encrypted data, but *without verification of its correctness*. On the other hand, the works on verifying computation seem focused on the case that the data is in the clear (an exception is [GKP+13] which we discuss in detail below).

The protocols for verifiable computation in [GGP10,CKV10,AIK10] use FHE as a tool to achieve verifiability. Almost as a by-product of the techniques, they turn out to achieve privacy of the data as well (the [GGP10] protocol also hides "most" of the function being computed – only the topology of the circuit computing is revealed [BHR12]). In [GGP10] a formal definition of all the properties needed by a verifiable computation scheme is given *including* input privacy. However, the definition in [GGP10] is in a very weak model in which the adversary is *not* allowed to issue *verification queries* to the client. In other words, a cheating server who observes the behavior of the client when an incorrect proof is presented is *not* an adversary considered by the definition in [GGP10].

There was a reason for this: as discussed in [GGP10], such a cheating server could in fact break the protocols in [GGP10,CKV10,AIK10] *completely* (i.e. both input privacy *and* output correctness would be compromised). We revisit this attack later in this section. For now we simply point out that [GGP10] defined only what they could achieve, and assumed that the acceptance/rejection bit of the client remains private and not learned by the adversary.

Somewhat surprisingly, no formal definition of *private and secure* outsourced computation *in the presence of verification queries* can be found in the literature. Our goal in this paper is to address this somewhat problematic state of affairs.

In particular we present the following:

1. An upgrade to the definition in [GGP10] that allows verification queries by the adversary. This is the strongest possible model in which we can define security.
2. A protocol based on FHE that achieves the above definition for arbitrary poly-time computations.

3. Some additional protocols for the computation of *ad-hoc* functions such as the computation of polynomials and linear combinations over encrypted data. These protocols do not use the power of FHE, and therefore are much more efficient than the generic approach. These tasks had been considered by [BGV11,LPJY13]. We point out that the protocol in [BGV11] becomes insecure in the presence of verification queries (a fact not noticed in that paper), while our protocols can be proven in the stronger security model where verification queries are allowed; moreover, the protocol suggested in [LPJY13] for verifiable computation of linear combinations over encrypted data is restrictive, since the linear combinations have to sit in a small range for the client to retrieve the correct result.
   Both our ad-hoc protocols are expressed in the same format as the general protocol, and are "function private", in the sense that the data defining the function to be computed is not disclosed to the server (see Section 2); moreover, our protocol for verifiable computation of linear combinations achieves both function privacy and input privacy. To the best of our knowledge, this is the first protocol to achieve both properties.

1.2 The Technical Issues

One might think that to achieve a fully secure (i.e. both private and correct) outsourced computation, it is sufficient to encrypt the data a with FHE and also run a verifiable computation for the function f. In other words the client would send $C_a = \mathsf{FHE}(a)$ and the server would return $C = \mathsf{FHE}(f(a))$ together with a proof π that the decryption of C is the correct value. The client would then decrypt C and accept only if the proof verifies it.

First of all, it is not immediately obvious how to do this for arbitrary computations in a way that the proof be efficiently verifiable (i.e. in quasi-linear time) by the verifier. But even more importantly this approach has no hope of achieving any meaningful notion of privacy (e.g. semantic security) for the encrypted data a. Indeed, suppose that the server knows a priori information about the data (at the extreme that the data a can assume only two possible values a_0 or a_1). Then the server would discard C_a and run the protocol on $C_0 = \mathsf{FHE}(a_0)$. If the client accepts, then the server knows that $a = a_0$; if the client rejects, then it must be that $a = a_1$. In other words the proof of correctness of the result can be used as a decryption oracle on the FHE: in order for this protocol to be secure the encryption should be secure against chosen-ciphertext attacks, but that is not possible if we also need it to be homomorphic.

The only way to maintain semantic security using this approach is to assume that the server/adversary is not given access to the client's decision bit regarding acceptance of the proof. In other words, security is achieved in a scheme in which the adversary is not allowed arbitrary verification queries on the protocol. This is of course a strong assumption on the behavior of the adversary and a very weak security model.

1.3 Other Prior Work

Generic Protocols. A different line of work to achieve private and verifiable outsourced computation was introduced by Goldwasser *et al.* in [GKP$^+$13] on *functional encryption (FE)*. An FE scheme is an encryption scheme with a single public key PK and several secret keys: given a function f there is a secret key SK_f associated with it. Given c, the encryption of a message m under PK, the owner of the secret key SK_f successfully decrypts c to $f(m)$ and learns no other information about m. The main result in [GKP$^+$13] is the construction of such an FE scheme for any function defined by a poly-size circuit.

In a passing remark in the Introduction the authors of [GKP$^+$13] show that such an FE scheme can be used to construct a VC that operates over encrypted inputs and is resistant to verification queries (in particular it is publicly verifiable – anybody can verify that the server operates correctly). The idea originates in [PRV12], for the case of attribute-based encryption (a subset of FE): to outsource the computation of a boolean (i.e. with output in $\{0,1\}$) function f, the client gives the server the secret key $SK_{\hat{f}}$ (in a potentially expensive preprocessing phase) for the function $\hat{f}$ defined as follows: $\hat{f}(r,x) = r$ if $f(x) = 1$ otherwise $\hat{f}(r,x) = \perp$. During the computation stage (when the client asks the server to compute $f(x)$), the client encrypts the pair (r,x) for a random value r and submits c to the server. If the server returns the correct value r, then the client has a "proof" that $f(x) = 1$. Note that

– this is a "one-sided" proof since the client cannot receive a proof that $f(x) = 0$ (the server can always refuse to answer when $f(x) = 1$) – this can be addressed by outsourcing both f and its complement;
– This scheme is privately verifiable, but can be made publicly verifiable by publicizing an "obfuscated" version of the random value r;
– This scheme does not reveal x, but reveals $f(x)$. In [GKP+13] they also briefly discuss a neat technique which hides the value of $f(x)$.

We point out that the above construction is only sketched in [GKP+13], together with an intuitive motivation for its security, but no formal definition or proof is given. When one formally analyzes the protocol in [GKP+13] in light of our definition the following issues come up:

– As discussed in [BSW11], a simulation-based definition of security for FE cannot be achieved in general in an "adaptive" model in which the adversary chooses the function f to be evaluated *after* seeing the ciphertext encoding x. It appears that this limitation might be inherited by the FE-based verifiable computation in [GKP+13], though the situation is far from clear. Our security definition is a game-based one, but the security proof might require the FE tool to be simulatable. A proof of the [GKP+13] protocol in this model is important, since in many cloud computing application the data might sit encrypted on a cloud server, before any processing is performed on it. In contrast, we can provably show that our proposed protocol does not have this limitation (see Section 2);
– Additionally, if one requires "standard" security against poly-time adversaries the protocol in [GKP+13] only achieves *selective security*, where the adversary has to commit in advance to the input value on which it wants to cheat[1] while our protocol achieves full security for poly-time adversaries when instantiated with the appropriate FHE and VC components.
– Finally, the scheme in [GKP+13] intrinsically works for *binary* functions. While this is sufficient in theory, in practice it requires the overhead of working bit-by-bit on the function output. Our protocol is more versatile and potentially more efficient, since it can handle arithmetic circuits if the underlying FHE and VC schemes can (several FHE schemes work over arithmetic circuits, and the QSP/QAP approach in [GGPR13] yields an efficient VC over arithmetic circuits)[2].

Ad-Hoc Protocols. The verification of the result *after* decryption is actually suggested in [BGV11] that proposes protocols for "ad-hoc" computations (for the evaluation of polynomials) and does not require the full power of FHE, but simply additively homomorphic encryption. In that protocol, the client encrypts the data a under an additively homomorphic encryption scheme Enc and receives back $C = \mathsf{Enc}(f(a))$ and a proof π. In order to verify the proof, the client must decrypt C, creating therefore the opportunity for the above attack (neither the attack nor the weak countermeasure discussed above are mentioned in [BGV11]).

The verification of the result *before* decryption is employed in [LPJY13] that proposes a protocol for the evaluation of linear computation over encrypted data, using additively homomorphic encryption, and makes use of structure preserving signatures on the ciphertexts group. The encryption scheme in that protocol, however, is used in such a way that the client cannot decrypt the result of the computation in general (the client would need to compute the discrete logarithm of the encoded linear combination), so the protocol allows efficient decryption only if the computation of linear combinations is in a small range.

1.4 Our Generic Protocol

Our protocol[3] assumes the existence of a FHE scheme, and also of an efficiently verifiable scheme to outsource the computation of a function f over data a, sent to the server in the clear (e.g. [GGPR13]).

[1] This is because both known constructions of FE [GKP+13,GGH+13] achieve only selective security against poly-time adversaries – full security can be obtained at the cost of assuming security against sub-exponential adversaries: a much stronger assumption.

[2] Another thing to consider is that the FE construction in [GKP+13] requires the client to work in the *depth* of the function f being outsourced. However, the new FE scheme in [GGH+13] based on indistinguishability obfuscation removes this limitation (since in their case ciphertexts are succinct and do not depend on the size of the circuit computing f in any way).

[3] We have never seen this protocol suggested anywhere in the literature, and we are not even aware of it as some sort of *folklore* theorem.

The protocol encrypts the data a with an FHE scheme: $C_a = \mathsf{FHE}(a)$. It then runs a protocol for the verifiable outsourcing *not of* f but of the function $F = \mathsf{Eval}_{\mathsf{FHE}}(f)$, i.e. the circuit that evaluates $C = \mathsf{FHE}(f(a))$ over C_a, which is now the input "in the clear".

In order to hide both f and a, we follow the usual approach of outsourcing the universal circuit which takes as input both f and a.

We note that this technique prevents the adversary from using the client as a decryption oracle for the FHE. Indeed if the verifiable computation component is secure even in the presence of verification queries (e.g. the QSP-based protocol in [GGPR13]) then the acceptance/rejection decision bit of the client is not linked to any decryption but just to the "correct" evaluation of the $\mathsf{Eval}_{\mathsf{FHE}}$ procedure over the ciphertexts (which we assume to be deterministic).

1.5 Protocols for computations over large datasets

In the last part of the paper we deal with the following problem. Let us assume that the data a consists of a large dataset $a = a_0, a_1, \ldots, a_d$, stored at the server with with some authentication information $t = t_0, t_1, \ldots, t_d$.

Polynomials Following [BGV11] the function f consists of $f(x) = A(x) = \sum_{i=0}^{d} a_i x^i \bmod p$, the computation of the polynomial of degree d defined by the a_i at the point x. In [BGV11] the following approach is proposed to efficiently verify this computation when the a_i are in the clear.

The authentication information stored by the client with the server is of the form $t_i = g^{c \cdot a_i + r_i}$ where $c \leftarrow \mathbb{F}_p$, $r_i \leftarrow \mathbb{F}_p$ is the ith coefficient of a polynomial $R(\cdot)$ of degree d and g is the generator of a cyclic group G of order p in which the discrete logarithm problem is hard. When queried on the point x the server returns $a = \sum_{i=0}^{d} a_i x^i$ and $t = \prod_{i=0}^{d} t_i^{x^i}$ and the client accepts a iff $t = g^{c \cdot a} g^{R(x)}$.

If $R(\cdot)$ is a random polynomial, then this is a secure delegation scheme. To avoid storing a large polynomial $R(\cdot)$ and the expensive ($O(N)$) computation of $g^{R(x)}$, the notion of *closed-form efficient* PRF is presented in [BGV11]. The value $\rho_i = g^{r_i}$ is defined as $PRF_K(i)$ for a special form of pseudo-random function PRF which has the property that the computation of $g^{R(x)}$ can be performed in $o(N)$ time by the party who knows the secret key K.

Since the values t_i reveal no information about the dataset a, in [BGV11] it is suggested that in order to protect the secrecy of a it is sufficient to encrypt it with an additively homomorphic encryption scheme E. No more details are provided in the paper, but we assume that this means setting $\alpha_i = \mathsf{Enc}(a_i)$ and using the homomorphism the server returns an encryption α of a (since a is simply a linear combination of the a_i's with weights x^i) and the value t as before. At that point the client would decrypt α to a' and check correctness as above. Clearly this approach is subject to the verification query attack, and would require hiding the client decision bit (i.e. forbidding verification queries – this attack is not noted in [BGV11]).

In Section 4.3 we construct a protocol that carries the basic ideas of [BGV11], has a fast verification phase, is function private, secure even under verification queries, and extends the computation to a polynomial with coefficients in an extension field $\mathbb{F}_{p^n}$.

Our solution is given by modifying the protocol in [BGV11] by assuming that the values t_i are authentications of the ciphertexts α_i, and that the verification check is carried out on the ciphertext α.

We need an additively homomorphic scheme (in which additions of plaintexts are correspond to additions of ciphertexts). Such encryption schemes exist under a variety of lattice-based assumptions, and for concreteness we use the scheme by Brakerski and Vaikuntanathan in [BV11].

Linear Combinations Another specialized protocol we propose works in case a client wants to perform linear combinations of the values in its dataset (which is a generalization of the above scenario). More specifically, the client wants to compute $\sum_{i=0}^{d} a_i \cdot w_i$ for some weights $w_0, \ldots, w_d \in \mathcal{M}$ – this setting is a generalization of the case of the evaluation of polynomials, for which $w_i = x^i$ for $x \in \mathbb{F}_p$.

In [LPJY13] the case of general linear combination is considered using *structure preserving* cryptography. Their scheme also uses additively homomorphic cryptography and does not require the client to decrypt the result a to perform the verification check. However, their scheme is limited in many other ways (for example,

in the specific instantiation they describe, which makes use of the encryption scheme by Boneh, Boyen, and Shacham [BBS04], the decryption of the value a by the client requires the computation of unknown discrete logarithms or an exhaustive search and therefore is efficient only for small a's, which limits the kind of linear combinations one can do, and the size d of the dataset).

General linear combinations in the model in which verification queries are not allowed can be easily achieved by a simpler version of the [BGV11] protocol where the value r_i is computed using a regular PRF (i.e. closed-form efficiency is not even required). The client stores $\alpha_i = \mathsf{Enc}(a_i)$ and the authentication information $t_i = c \cdot a_i + r_i$ where $c \leftarrow \mathcal{M}$, and $r_i = \mathsf{PRF}_k(i)$. When the client provides the weights w_i, the server computes α such that $\mathsf{Dec}(\alpha) = \sum_{i=0}^{d} a_i \cdot w_i$ (using the homomorphic properties of Enc) and $t = \sum_{i=0}^{d} t_i \cdot w_i$. The client decrypts α and accepts if and only if $t = c \cdot a + \sum_{i=0}^{d} r_i \cdot w_i$.

To achieve function privacy against verification queries, we change the authentication tags to be $t_i = c \cdot \alpha_i + r_i$. Now, if we use the [BV11] encryption scheme we can verify the correct evaluation of α before decrypting. Moreover, by using the variant of [BV11] that allows the evaluation of a single multiplication on encrypted data, the client can even encrypt the weights w_i, i.e. hide the linear combination to be computed and achieve input privacy as well. In this case, the client provides $\omega_i = \mathsf{Enc}(w_i)$ to the server, which uses the single multiplication homomorphic property to compute α such that $\mathsf{Dec}(\alpha) = a$ and τ such that $\mathsf{Dec}(\tau) = t$. As we describe in detail in Section 4.2 the verification check can still be performed before decrypting, and therefore provides no useful decryption query to the adversary.

We note that the client complexity here depends on the size of the weight set and on the level of information the client wants to hide. If the weights were sent in the clear, the client would have paid $O(\widetilde{d})$ complexity where $\widetilde{d} \leq d$ is the number of non-zero weights. If the weights are encrypted, the client has two choices: he can still pay $O(\widetilde{d})$ by simply sending the non-zero weights encrypted (and their indices). In this case, however, the client reveals the indices used in the linear combination (though not their values). If the client wants to completely hide the weights then it must pay $O(d)$ to send a ciphertext ω_i for each index.

2 Problem Definition

We work in the amortized model of [GGP10] where the client runs a one-time expensive phase to "outsource" the function f to the server (this phase can cost as much as the computation of f). Later the client queries the server on (an encrypted form of) input x and receives back (an encryption of) the value $f(x)$ and a proof of its correctness: this phase should be efficient for the client (ideally linear in $|x| + |f(x)|$).

In [GGP10] the authors give a definition that includes both security (i.e. the client only accepts correct outputs) and privacy (i.e. the client's input x is semantically secure) but does *not* allow verification queries (since the [GGP10] protocol does not tolerate them). In this Section we "upgrade" the definition by adding verification queries to it.

A *verifiable computation scheme* $\mathcal{VC} = (\mathbf{KeyGen}, \mathbf{ProbGen}, \mathbf{Compute}, \mathbf{Verify})$ consists of the four algorithms defined below.

1. $\mathbf{KeyGen}(f, \lambda) \rightarrow (PK, SK)$: 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. $\mathbf{ProbGen}_{SK}(x) \rightarrow (\sigma_x, \tau_x)$: The *problem generation* algorithm uses the secret key SK to encode the function input x as a public value σ_x which is given to the server to compute with, and a secret value τ_x which is kept private by the client.
3. $\mathbf{Compute}_{PK}(\sigma_x) \rightarrow \sigma_y$: 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. $\mathbf{Verify}_{SK}(\tau_x, \sigma_y) \rightarrow (acc, y)$: Using the secret key SK and the secret "decoding" τ_x, the *verification* algorithm converts the server's encoded output into a bit acc and a string y. If $acc = 1$ we say the client accepts $y = f(x)$, if $acc = 0$ we say the client rejects.

We now recall the three main properties defined in [GGP10] for a verifiable computation scheme: correctness, security and privacy but we define them in the *presence* of verification queries by the adversary.

Also, we introduce another definition, that encompasses function privacy, that is the ability of a scheme to hide from the server the function that it needs to compute.

A scheme is correct if the problem generation algorithm produces values that allow an honest server to compute values that will verify successfully and correspond to the evaluation of f on those inputs. More formally:

Definition 1 (Correctness). *A verifiable computation scheme $\mathcal{VC}$ is* correct *if for any choice of function f, the key generation algorithm produces keys $(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$ such that, for all x the domain of f, if $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{SK}(x)$ and $\sigma_y \leftarrow \mathbf{Compute}_{PK}(\sigma_x)$ then $(1, y = f(x)) \leftarrow \mathbf{Verify}_{SK}(\tau_x, \sigma_y)$.*

Intuitively, a verifiable computation scheme is secure if a malicious server cannot persuade the verification algorithm to accept an incorrect output. Below, we formalize this intuition with an experiment, where $\mathsf{poly}(\cdot)$ is a polynomial.

Note that the adversary A is given access to queries to the oracle **PVerify** which is defined as

$$\mathbf{PVerify}(\tau, \sigma) = acc \text{ if and only if } \mathbf{Verify}(\tau, \sigma) = (acc, y)$$

In other words, **PVerify** is the public acceptance/rejection bit which results from a verification query.

Experiment $\mathbf{Exp}_A^{Verif}[\mathcal{VC}, f, \lambda]$
$(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$;
For $i = 1, \ldots, \ell = \mathsf{poly}(\lambda)$;
$x_i \leftarrow A^{\mathbf{PVerify}(\tau_j, \cdot)}(PK, x_1, \sigma_1, \ldots, x_{i-1}, \sigma_{i-1})$;
$(\sigma_i, \tau_i) \leftarrow \mathbf{ProbGen}_{SK}(x_i)$;
$(i, \hat{\sigma}_y) \leftarrow A^{\mathbf{PVerify}(\tau_j, \cdot)}(PK, x_1, \sigma_1, \ldots, x_\ell, \sigma_\ell)$;
$(\hat{acc}, \hat{y}) \leftarrow \mathbf{Verify}_{SK}(\tau_i, \hat{\sigma}_y)$
If $\hat{acc} = 1$ and $\hat{y} \neq f(x_i)$, output '1', else '0';

Essentially, the adversary is given oracle access to generate the encoding of multiple problem instances, and to check the response of the client on arbitrary "encodings". The adversary succeeds if it produces an output that convinces the verification algorithm to accept on the wrong output value for a given input value. We can now define the security of the system based on the adversary's success in the above experiment.

Definition 2 (Security). *A verifiable computation scheme $\mathcal{VC}$ is* secure *for a function f, if for any adversary A running in probabilistic polynomial time,*

$$\Pr[\mathbf{Exp}_A^{Verif}[\mathcal{VC}, f, \lambda] = 1] \leq \mathsf{negl}(\lambda). \tag{2.1}$$

We say that $\mathcal{VC}$ is secure if it is secure for every function f.

Input privacy is defined based on a typical indistinguishability argument that guarantees that *no* information about the inputs is leaked. Input privacy, of course, immediately yields output privacy.

Intuitively, a verifiable computation scheme is *private* when the public outputs of the problem generation algorithm **ProbGen** over two different inputs are indistinguishable; i.e. nobody can decide which encoding is the correct one for a given input. More formally consider the following experiment: the adversary is given the public key for the scheme and selects two inputs x_0, x_1. He is then given the encoding of a randomly selected one of the two inputs and must guess which one was encoded. During this process the adversary is allowed to request the encoding of any input he desires, and also is allowed to make verification queries on any input. The experiment is described below[4]. The oracle $\mathbf{PProbGen}_{SK}(x)$ calls $\mathbf{ProbGen}_{SK}(x)$ to obtain (σ_x, τ_x) and returns only the public part σ_x.

[4] In the definition when we write that $A^{\mathbf{PVerify}}$ we mean that A is allowed to query $\mathbf{PVerify}(\tau, \cdot)$ where τ is the secret encoding of any of the queries made in **PProbGen** *and* τ_b

Experiment $\mathbf{Exp}_A^{Priv}[\mathcal{VC}, f, \lambda]$
$(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$;
$(x_0, x_1) \leftarrow A^{\mathbf{PVerify}, \mathbf{PProbGen}_{SK}(\cdot)}(PK)$
$(\sigma_0, \tau_0) \leftarrow \mathbf{ProbGen}_{SK}(x_0)$;
$(\sigma_1, \tau_1) \leftarrow \mathbf{ProbGen}_{SK}(x_1)$;
$b \leftarrow \{0, 1\}$;
$\hat{b} \leftarrow A^{\mathbf{PVerify}, \mathbf{PProbGen}_{SK}(\cdot)}(PK, x_0, x_1, \sigma_b)$
If $\hat{b} = b$, output '1', else '0';

Definition 3 (Privacy). *A verifiable computation scheme $\mathcal{VC}$ is* private *for a function f, if for any adversary A running in probabilistic polynomial time,*

$$\Pr[\mathbf{Exp}_A^{Priv}[\mathcal{VC}, f, \lambda] = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda). \tag{2.2}$$

Function privacy is the requirement that the public key PK, sampled via $(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$, does not leak information on the encoded function f, even after a polynomial amount of runs of $\mathbf{ProbGen}_{SK}$ on adversarially chosen inputs. More formally, we define define function privacy based on an indistinguishability experiment as follows

Experiment $\mathbf{Exp}_A^{FPriv}[\mathcal{VC}, \lambda]$
$(f_0, f_1) \leftarrow A(\lambda)$;
$b \leftarrow \{0, 1\}$;
$(PK, SK) \leftarrow \mathbf{KeyGen}(f_b, \lambda)$;
For $i = 1, \ldots, \ell = \mathsf{poly}(\lambda)$;
$x_i \leftarrow A^{\mathbf{PVerify}(\tau_j, \cdot)}(PK, x_1, \sigma_1, \ldots, x_{i-1}, \sigma_{i-1})$;
$(\sigma_i, \tau_i) \leftarrow \mathbf{ProbGen}_{SK}(x_i)$;
$\hat{b} \leftarrow A^{\mathbf{PVerify}(\tau_j, \cdot)}(PK, x_1, \sigma_1, \ldots, x_\ell, \sigma_\ell)$;
If $\hat{b} = b$, output '1', else '0';

Definition 4 (Function Privacy). *A verifiable computation scheme $\mathcal{VC}$ is* function private*, if for any adversary A running in probabilistic polynomial time,*

$$\Pr[\mathbf{Exp}_A^{FPriv}[\mathcal{VC}, \lambda] = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda). \tag{2.3}$$

The final condition we require from a verifiable computation scheme is that the time to encode the input and verify the output must be smaller than the time to compute the function from scratch.

Definition 5 (Outsourceable). *A $\mathcal{VC}$ can be outsourced if it permits efficient generation and efficient verification. This implies that for any x and any σ_y, the time required for $\mathbf{ProbGen}_{SK}(x)$ plus the time required for $\mathbf{Verify}(\sigma_y)$ is $o(T)$, where T is the time required to compute $f(x)$.*

2.1 Adaptive Security

Here we introduce the notion of adaptive security for a verifiable computation scheme, and show that our constructions fulfill this definition. Intuitively, an adaptively secure scheme is a scheme that is secure even if the adversary chooses f after having seen many "encodings" σ_x for chosen values x. At first sight, this property is non-trivial to achieve, since not every scheme allows σ_x to be computed before choosing f (in particular schemes based on FE such as [GKP+13]). This observation leads us to first define a refined class of schemes, for which adaptivity is not ruled out by this restriction, and then proceed with the actual definition of adaptivity.

Definition 6 (Split Scheme). *Let $\mathcal{VC} = (\mathbf{KeyGen}, \mathbf{ProbGen}, \mathbf{Compute}, \mathbf{Verify})$ be a verifiable computation scheme. We say that $\mathcal{VC}$ is a* split scheme *if the following conditions hold:*

- *There exist PPT algorithms* $\mathbf{KeyGen}^E(\lambda)$ *and* $\mathbf{KeyGen}^V(f,\lambda)$ *such that if* $(PK,SK) \leftarrow \mathbf{KeyGen}(f,\lambda)$ *then* $PK = (PK_E, PK_V)$ *and* $SK = (SK_E, SK_V)$ *such that* $\mathbf{KeyGen}^E(\lambda) \rightarrow (PK_E, SK_E)$ *and* $\mathbf{KeyGen}^V(f,\lambda) \rightarrow (PK_V, SK_V)$*;*
- *There exist PPT algorithms* $\mathbf{ProbGen}^E_{SK_E}(x)$ *and* $\mathbf{ProbGen}^V_{SK_V}(x)$ *such that if* $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{SK_E,SK_V}(x)$ *then* $\sigma_x = \sigma_x^E \leftarrow \mathbf{ProbGen}^E_{SK_E}(x)$ *and* $\tau_x = \tau_x^V \leftarrow \mathbf{ProbGen}^V_{SK_V}(x)$

Notice that for a split scheme one can generate valid values σ_x for any function f to be delegated before knowing f, since σ_x is independent of f. This can be done by running $(PK_E, SK_E) \leftarrow \mathbf{KeyGen}^E(\lambda)$, and setting $\sigma_x \leftarrow \mathbf{ProbGen}^E_{SK_E}(x)$ before knowing f. The validity of this encoding applies for all keys $(PK,SK) = (PK_E, PK_V, SK_E, SK_V)$ where $(PK_V, SK_V) \leftarrow \mathbf{KeyGen}^V(f,\lambda)$ for *any* f.

We can now describe the experiment that will be used to define adaptive security for split schemes:

Experiment $\mathbf{Exp}_A^{Adap-Verif}[\mathcal{VC},\lambda]$
$(PK_E, SK_E) \leftarrow \mathbf{KeyGen}^E(\lambda)$;
For $i = 1,\ldots,\ell' = \mathsf{poly}'(\lambda)$:
$x'_i \leftarrow A(PK_E, x'_1, \sigma'_1, \ldots, x'_{i-1}, \sigma'_{i-1})$;
$\sigma'_i \leftarrow \mathbf{ProbGen}^E_{SK_E}(x)$;
$f \leftarrow A(x'_1, \sigma'_1, \ldots, x'_{\ell'}, \sigma'_{\ell'})$;
$(PK_V, SK_V) \leftarrow \mathbf{KeyGen}^V(f,\lambda)$;
$(PK,SK) \leftarrow (PK_E, PK_V, SK_E, SK_V)$;
For $i = 1,\ldots,\ell = \mathsf{poly}(\lambda)$:
$x_i \leftarrow A^{\mathbf{PVerify}(\tau_j,\cdot)}(PK, x_1, \sigma_1, \ldots, x_{i-1}, \sigma_{i-1})$;
$(\sigma_i, \tau_i) \leftarrow \mathbf{ProbGen}_{SK}(x_i)$;
$(i, \hat{\sigma}_y) \leftarrow A^{\mathbf{PVerify}(\tau_j,\cdot)}(PK, x_1, \sigma_1, \ldots, x_\ell, \sigma_\ell)$;
$(\hat{acc}, \hat{y}) \leftarrow \mathbf{Verify}_{SK}(\tau_i, \hat{\sigma}_y)$;
If $\hat{acc} = 1$ and $\hat{y} \neq f(x_i)$, output '1', else '0'.

Definition 7 (Adaptive Security). *A split scheme* $\mathcal{VC}$ *is* adaptively secure*, if for any adversary* A *running in probabilistic polynomial time,*

$$\Pr[\mathbf{Exp}_A^{Adap-Verif}[\mathcal{VC},\lambda] = 1] \leq \mathsf{negl}(\lambda). \tag{2.4}$$

We point out that a particular instantiation of our generic scheme is split: consider the case in which the VC component of our generic solution is implemented using a SNARK (e.g. any of the protocols in [BCCT12b,GGPR13,PHGR13,BSCG+13]). In those protocols basically τ_x can be "empty" (as many of those protocols are publicly verifiable). As a consequence our generic protocol in this case is split (the encoding of x is just an FHE encryption of x, the secret key SK the decryption key for the FHE scheme).

Once we have a split scheme, our generic scheme can be easily proven to be adaptively secure along the lines of the proof of Theorem 1.

3 A Generic Solution

In this section we describe our generic solution to outsource computation over encrypted data. As we discussed in the introduction we assume the existence of a FHE scheme, with a deterministic evaluation procedure (all known schemes have this property). We also need an efficiently verifiable scheme to outsource the computation of a function f over data a, sent to the server in the clear, which is secure even in the presence of verification queries. The latter can be achieved using a variety of schemes, e.g. [GGPR13,GLR11,BCCT12a,BCCT12b]

The protocol encrypts the data a with an FHE scheme: $C_a = \mathsf{FHE}(a)$. It then runs a protocol for the verifiable outsourcing *not of* f but of the function $F(\cdot) = \mathsf{Eval}_{\mathsf{FHE}}(f,\cdot)$, which is a circuit that takes C_a as input and outputs an element C that decodes to $f(a)$.

We note that this approach prevents the adversary from using the client as a decryption oracle for the FHE. Indeed if the verifiable computation component is secure even in the presence of verification queries then the acceptance/rejection decision bit of the client is not linked to any decryption but just to the "correct" evaluation of the $\mathsf{Eval}_{\mathsf{FHE}}$ procedure over the ciphertexts (which we assume to be deterministic).

3.1 Homomorphic Encryption

A *fully homomorphic (public-key) encryption* (FHE) scheme is a quadruple of PPT algorithms $\mathsf{HE} = (\mathsf{HE.KeyGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$ defined as follows.

- $\mathsf{HE.KeyGen}(\lambda) \to (\mathsf{pk}, \mathsf{evk}, \mathsf{sk})$: Outputs a public encryption key pk, a public evaluation key evk and a secret decryption key sk.
- $\mathsf{HE.Enc}_{\mathsf{pk}}(b) \to c$: Encrypts a bit $b \in \{0,1\}$ under public key pk. Outputs ciphertext c.
- $\mathsf{HE.Dec}_{\mathsf{sk}}(c) \to b$: Decrypts ciphertext c using sk to a plaintext bit $b \in \{0,1\}$.
- $\mathsf{HE.Eval}_{\mathsf{evk}}(g, c_1, \ldots, c_t) \to c^*$: The *deterministic evaluation algorithm* takes the evaluation key evk, a boolean circuit $g : \{0,1\}^t \to \{0,1\}$, and a set of t ciphertexts $c_1, \ldots, c_t$. It outputs the result ciphertext c^*.

An FHE should also satisfy the following properties.

Encryption Correctness. For all $b \in \{0,1\}$ we have:

$$\Pr\left[\mathsf{HE.Dec}_{\mathsf{sk}}(\mathsf{HE.Enc}_{\mathsf{pk}}(b)) = b \mid (\mathsf{pk}, \mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{HE.KeyGen}(\lambda)\right] = 1$$

Evaluation Correctness. For any $(\mathsf{pk}, \mathsf{evk}, \mathsf{sk})$ in the support of $\mathsf{HE.KeyGen}(\lambda)$, any ciphertexts $c_1, \ldots, c_t$ such that $\mathsf{HE.Dec}_{\mathsf{sk}}(c_i) = b_i \in \{0,1\}$, and any circuit $g : \{0,1\}^t \to \{0,1\}$ we have

$$\mathsf{HE.Dec}_{\mathsf{sk}}(\mathsf{HE.Eval}_{\mathsf{evk}}(g, c_1, \ldots, c_t)) = g(b_1, \ldots, b_t).$$

Succinctness. We require that the ciphertext-size is always bounded by some fixed polynomial in the security parameter, and is independent of the size of the evaluated circuit or the number of inputs it takes. That is, there exists some polynomial $p(\cdot)$ such that, for any $(\mathsf{pk}, \mathsf{evk}, \mathsf{sk})$ in the support of $\mathsf{HE.KeyGen}(\lambda)$, the output-size of $\mathsf{HE.Enc}_{\mathsf{pk}}(\cdot)$ and of $\mathsf{Eval}_{\mathsf{evk}}(\cdot)$ is bounded by $p(n)$, for any choice of their inputs.

Semantic Security. Lastly, an FHE should satisfy the standard notion of *semantic security* for public-key encryption, where we consider the evaluation key evk as a part of the public key. That is, for any PPT attacker A we have:

$$\left|\Pr\left[A(\lambda, \mathsf{pk}, \mathsf{evk}, c_0) = 1\right] - \Pr\left[A(\lambda, \mathsf{pk}, \mathsf{evk}, c_1) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

where the probability is over $(\mathsf{pk}, \mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda), \{c_b \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(b)\}_{b \in \{0,1\}}$, and the coins of A.

3.2 The Scheme

Let $\mathsf{HE} = (\mathsf{HE.KeyGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$ be an FHE scheme as defined above.
Also let $\mathcal{VC} = (\mathbf{KeyGen}, \mathbf{ProbGen}, \mathbf{Compute}, \mathbf{Verify})$ be a verifiable computation scheme which is correct, secure, and outsourceable, as defined in Section 2. In particular note that $\mathcal{VC}$ needs not to be private, and that the security is guaranteed in the presence of verification queries.

We now describe a new verifiable computation scheme $\mathcal{PVC}$ (for private VC) which uses the above two tools. $\mathcal{PVC} = (\mathbf{PKeyGen}, \mathbf{PProbGen}, \mathbf{PCompute}, \mathbf{PVerify})$ defined as follows.

1. $\mathbf{PKeyGen}(f, \lambda) \to (PK_P, SK_P)$: it first runs $\mathsf{HE.KeyGen}(\lambda)$ to generate $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk})$ for the FHE scheme HE.
   Let $eval_f$ be the function that takes as input $\mathsf{HE.Enc}_{\mathsf{pk}}(x)$ and outputs $\mathsf{HE.Enc}_{\mathsf{pk}}(f(x))$. Given $\mathsf{pk}, \mathsf{evk}$ this function is efficiently computable (given that HE is an FHE), therefore we can run $\mathbf{KeyGen}(eval_f, \lambda)$ to generate the PK, SK for $\mathcal{VC}$.
   It sets $PK_P = (PK, \mathsf{pk}, \mathsf{evk})$ and $SK_P = (PK_P, SK, \mathsf{sk})$.
2. $\mathbf{PProbGen}_{SK_P}(x) \to (\sigma_x, \tau_x)$: It first computes $C_x = \mathsf{HE.Enc}_{\mathsf{pk}}(x)$, and then it runs $\mathbf{ProbGen}_{SK}(C_x)$ to get (σ_x, τ_x)
3. $\mathbf{PCompute}_{PK_P}(\sigma_x) \to \sigma_y$: It runs $\mathbf{Compute}_{PK}(\sigma_x)$ to compute σ_y. Note that since we are outsourcing the function $eval_f$ the value σ_y should be an encoding of $C_y = \mathsf{HE.Eval}_{\mathsf{evk}}(f, C_x) = \mathsf{HE.Enc}_{\mathsf{pk}}(y = f(x))$

4. $\mathbf{PVerify}_{SK_P}(\tau_x, \sigma_y) \rightarrow (acc, y)$: Run $\mathbf{Verify}_{SK}(\tau_x, \sigma_y)$ to get (acc, C). If $acc = 0$ then reject. If $acc = 1$ then decrypt $y = \mathsf{HE.Dec}_{\mathsf{sk}}(C)$.

Theorem 1. *If* HE *is a semantically secure FHE, and* $\mathcal{VC}$ *is a correct, secure and outsourceable verifiable computation scheme (as defined in Section 2 i.e. in the presence of verification queries), then* $\mathcal{PVC}$ *is a correct, secure, outsourceable and* private *verifiable computation scheme (as defined in Section 2 i.e. in the presence of verification queries)*

Proof. **(Sketch)** Correctness of $\mathcal{PVC}$ follows from the correctness of HE and $\mathcal{VC}$. Similarly the fact that $\mathcal{VC}$ is secure and outsourceable implies the same properties for $\mathcal{PVC}$.

The one thing to argue then is privacy for $\mathcal{PVC}$, and we prove that from the semantic security of HE. In other words we show that if an adversary A can learn any information about the input x to **PProbGen** in $\mathcal{PVC}$ then we can use A to break the semantic security of HE.

Let us assume then that there exists A, f such that

$$\Pr[\mathbf{Exp}_A^{Priv}[\mathcal{VC}, f, \lambda] = 1] \geq \zeta$$

where $\zeta = \zeta(\lambda)$ is non-negligible in λ. We build a simulator S which is allowed to query A as an oracle and such that

$$\left|\Pr\left[S^A(\lambda, \mathsf{pk}, \mathsf{evk}, c_0) = 1\right] - \Pr\left[S^A(\lambda, \mathsf{pk}, \mathsf{evk}, c_1) = 1\right]\right| \geq \zeta$$

where the probability is over $(\mathsf{pk}, \mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda), \{c_b \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(b)\}_{b\in\{0,1\}}$, and the coins of S.

On input $\mathsf{pk}, \mathsf{evk}, c_b$ the simulator S runs as follows:

1. Run $\mathbf{KeyGen}(eval_f, \lambda)$ to generate the PK, SK for $\mathcal{VC}$ on $eval_f$. It sets $PK_P = (PK, \mathsf{pk}, \mathsf{evk})$.
2. Run A on PK_P. Remember that in this step A is allowed two types of queries:
   - Queries to **PProbGen** which S can answer since it knows the public key pk of HE and the secret key SK of $\mathcal{VC}$
   - Queries to **PVerify** which S can also answer since it knows the secret key SK of $\mathcal{VC}$. Remember that **PVerify** returns only the acceptance/rejection bit acc which S can calculate using only SK. The secret key sk of HE (which S does not have) is not needed to answer these queries.
3. At some point A outputs two inputs $x_0 \neq x_1$. Let us assume for now that x_0 and x_1 differ in a single bit, for example the first. We will show later how to get rid of this assumption by a standard hybrid argument. So let's assume that x_0 starts with 0 and x_1 starts with 1.
   S will construct C_{x_b} by concatenating c_b with the encryptions of all the other bits (which S can compute using pk).
   S will finish to run **ProbGen** on x_b, to compute σ_{x_b}, τ_{x_b} and returns σ_{x_b} to A. This part requires only knowledge of SK so S can simulate it.
4. A will continue running and making queries to **PProbGen** and **PVerify** which the simulator will be able to answer as above.
5. Finally A outputs a bit $\hat{b}$ which is equal to b with probability $1/2 + \zeta$. S outputs the same bit, and therefore S will also be correct with probability $1/2 + \zeta$.

To finish the proof we need to remove the assumption we made on the behavior of A in step 3. Let us assume that $x_0, x_1 \in \{0,1\}^n$ and set $x_b = [x_{b,1}x_{b,2}\ldots x_{b,n}]$. Define the string $x^{(j)} = [x_{1,1}x_{1,2}\ldots x_{1,j}x_{0,j+1}\ldots x_{0,n}]$, so $x^{(0)} = x_0$ and $x^{(n)} = x_1$.

If A distinguishes between x_0 and x_1 with advantage ζ then by a standard hybrid argument there must exists a j such that A distinguishes between $x^{(j-1)}$ and $x^{(j)}$ with advantage at least ζ/n. Notice that in order to be the case we must have $x_{0,j} \neq x_{1,j}$. The proof then continues with S guessing the bit j and placing the challenge ciphertext c_b in position j of the ciphertext C_{x_b} sent to A. S will still guess the correct bit with non-negligible advantage.

3.3 Hiding the Function

We point out that by outsourcing the universal circuit computing functions of a given size, we can hide not only the input data, but also the function being computed, so our scheme can be compiled into one which is also function private, according to definition 4, and without a significant loss in performance.

4 Ad-Hoc Solutions

In this section we describe two more efficient solutions, in case the function to evaluate on the dataset is of a specific form (linear functions and evaluations of polynomials).

4.1 Tools

For our concrete construction we use the somewhat homomorphic encryption scheme by Brakerski and Vaikuntanathan described in [BV11], based on the hardness of the polynomial learning with error problem, and we specialize it to evaluate circuits of multiplicative depth one on encrypted data. The specifications of the encryption scheme are as follows:

- $\mathsf{BV.ParamGen}(\lambda)$: The message space $\mathcal{M}$ is the ring $R_p := \mathbb{F}_p[X]/F(X)$, where $F(X)$ is a polynomial in $\mathbb{F}_p[X]$ of degree n. In the following we choose p and $F(X)$ such that $F(X)$ is irreducible in $\mathbb{F}_p[X]$, so that R_p is isomorphic to $\mathbb{F}_{p^n}$ (as a field). Operations in $\mathcal{M}$ are denoted with $+$ for addition and $\cdot$ for polynomial multiplication modulo $F(X)$. The homomorphic properties of the scheme are over R_p. We choose to represent elements in $\mathcal{M}$ as elements in $\mathbb{Z}^n$ with infinity norm bounded by $p/2$.
  The ciphertext space is determined as follows: pick a large integer q which is co-prime to p (for the choice of q we rely on the analysis done in [BV11], Section 2), and define $R_q := \mathbb{F}_q[X]/F(X)$. In practice, we consider plaintexts to be in R_q, when performing encryption and decryption (so computing arbitrary circuits on encrypted data may cause overflows and yield incorrect decryption). The ciphertext space $\mathcal{C}$ is defined as $\mathcal{C} := R_q^3$, and operations on ciphertexts are defined as follows:

$$(a_0, a_1, a_2) + (b_0, b_1, b_2) := (a_0 + b_0, a_1 + b_1, a_2 + b_2),$$
$$(a_0, a_1, 0) \boxtimes (b_0, b_1, 0) := (a_0 \cdot b_0, a_1 \cdot b_0 + b_1 \cdot a_0, a_1 \cdot b_1).$$

  Notice that $\boxtimes$ (multiplication of ciphertexts) is defined only for ciphertexts where the last entry equals zero.
  Finally, the algorithm defines $D_{Z^n,\sigma}$, the discrete Gaussian with parameter σ: it is the random variable over $\mathbb{Z}^n$ obtained from sampling $x \in \mathbb{R}^n$ with probability $e^{-\pi \cdot \|x\|_2/\sigma^2}$ and then rounding at the nearest lattice point. Again, we refer to [BV11] for the specific choice of σ. In the following, we assume that the parameters generated here are inputs of any subsequent algorithm.
- $\mathsf{BV.KeyGen}() \to (\mathsf{pk}, \mathsf{sk})$: The key-generation algorithm samples $a \leftarrow R_q$, and $s, e \leftarrow D_{\mathbb{Z}^n,\sigma}$. Considering s and e as elements in R_q, it computes $b \leftarrow a \cdot s + p \cdot e$, and sets $\mathsf{sk} \leftarrow s$ and $\mathsf{pk} \leftarrow (a, b)$.
- $\mathsf{BV.Enc}_{\mathsf{pk}}(m, r) \to (c_0, c_1, c_2)$: Given $m \in R_p$, and $r \leftarrow \left(D_{\mathbb{Z}^{3\cdot n},\sigma} \bmod q\right)$, the message m is parsed as an element in R_q with infinity norm bounded by $p/2$, and the randomness r is parsed as $r = (u, v, w) \in R_q^3$. The output is $c = (c_0, c_1, 0) \in R_q^3$, where $c_0 \leftarrow b \cdot v + p \cdot w + m$ and $c_1 \leftarrow a \cdot v + p \cdot u$.
- $\mathsf{BV.Dec}_{\mathsf{sk}}(c_0, c_1, c_2) \to t \bmod p$: The algorithm computes an element $t \in R_q$ as $t \leftarrow c_0 - s \cdot c_1 - s^2 \cdot c_2$. The output is then $t \bmod p$, which is interpreted as an element in R_p.

Lemma 1 ([BV11], Lemma 4; and [BV11], Theorem 2).
For $D = 1, 2$, the public-key encryption scheme $(\mathsf{BV.ParamGen}, \mathsf{BV.KeyGen}, \mathsf{BV.Enc}, \mathsf{BV.Dec})$*, specified above, is semantically secure under the PLWE assumption, and it allows the computation of any polynomial f of degree D such that*

$$\|f\|_\infty \cdot \left(p \cdot \sigma \cdot n^{1.5}\right)^D \leq q/2.$$

We also assume the existence of a keyed pseudorandom function PRF with image in R_q.

4.2 Linear Combinations

We are now ready to show a scheme for delegated computation on encrypted data, in case the secret function to be computed is linear and over R_p, i.e. f is described by values $a_0, \ldots a_d \in R_p$, the input is represented by weights $w_0, \ldots, w_d \in R_p$, and the output is $\sum_{i=0}^{d} a_i \cdot w_i$.

The scheme $\mathcal{VC}_{\mathsf{LC}}$ is specified as follows:

1. **KeyGen**$(f = a_0, \dots a_d, \lambda) \to (PK, SK)$:
   - it runs $\mathsf{BV.ParamGen}(\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{BV.KeyGen}()$,
   - it samples a uniform MAC key $c \leftarrow R_q$, and a key k for PRF,
   - it computes $r_i \leftarrow \mathsf{PRF}_k(2 \cdot i)$, $s_i \leftarrow \mathsf{PRF}_k(2 \cdot i + 1)$, it defines $\rho_i \leftarrow (r_i, s_i, 0)$,
   - it computes $\alpha_i \leftarrow \mathsf{BV.Enc}_{\mathsf{pk}}(a_i)$, and $\tau_i \leftarrow c \cdot \alpha_i + \rho_i$,
   - it sets $PK = (\mathsf{pk}, \alpha_0, \tau_0, \dots, \alpha_d, \tau_d)$ and $SK = (\mathsf{pk}, \mathsf{sk}, c, k)$.
2. **ProbGen**$_{SK}(x = w_0, \dots, w_d) \to \sigma_x, \tau_x$: for $i = 0, \dots, d$
   - if σ_x, τ_x are not yet defined, it sets σ_x as a vector with zero entries, and $\tau_x \leftarrow 0 \in R_q^3$,
   - it computes $\omega_i \leftarrow \mathsf{BV.Enc}_{\mathsf{pk}}(w_i)$,
   - it computes $r_i \leftarrow \mathsf{PRF}_k(2 \cdot i)$, $s_i \leftarrow \mathsf{PRF}_k(2 \cdot i + 1)$, it defines $\rho_i \leftarrow (r_i, s_i, 0)$, as in the keygen algorithm,
   - it sets $\sigma_x = \sigma_x, \omega_i$,
   - it sets $\tau_x = \tau_x + \rho_i \boxtimes \omega_i$.
3. **Compute**$_{PK}(\sigma_x = \omega_0, \dots, \omega_d) \to \sigma_y$:
   - it computes $\alpha \leftarrow \sum_{i=0}^{d} \alpha_i \boxtimes \omega_i$, and $\tau \leftarrow \sum_{i=0}^{d} \tau_i \boxtimes \omega_i$,
   - sets $\sigma_y = \alpha, \tau$
4. **Verify**$_{SK}(\tau_x, \sigma_y = \alpha, \tau) \to (acc, a')$: If $\tau \neq c \cdot \alpha + \tau_x$, it rejects. Otherwise, it accepts and computes $a' \leftarrow \mathsf{BV.Dec}_{\mathsf{sk}}(\alpha)$.

Theorem 2. *The scheme $\mathcal{VC}_{\mathsf{LC}}$ is correct, adaptively secure, function private, input private.*

Proof. For correctness, if both the client and the server are honest, the client accepts; using the fact that $\boxtimes$ distributes over $+$ in the co-domain of $\mathsf{BV.Enc}_{\mathsf{pk}}$, the following holds:

$$\tau = \sum_{i=0}^{d} \tau_i \boxtimes \omega_i = \sum_{i=0}^{d} (c \cdot \alpha_i + \rho_i) \boxtimes \omega_i = \sum_{i=0}^{d} c \cdot \alpha_i \boxtimes \omega_i + \sum_{i=0}^{d} \rho_i \boxtimes \omega_i = c \cdot \alpha + \tau_x,$$

so the check at the verification step passes. Moreover, the output a' is the desired one, since it equals:

$$\mathsf{BV.Dec}_{\mathsf{sk}}(\alpha) = \mathsf{BV.Dec}_{\mathsf{sk}}\left(\sum_{i=0}^{d} \alpha_i \boxtimes \omega_i\right) = \mathsf{BV.Dec}_{\mathsf{sk}}\left(\sum_{i=0}^{d} \mathsf{BV.Enc}_{\mathsf{pk}}(a_i) \boxtimes \mathsf{BV.Enc}_{\mathsf{pk}}(w_i)\right) = \sum_{i=0}^{d} a_i \cdot w_i.$$

For adaptive security, we show firstly that the scheme is split: define $PK_E = SK_E = \mathsf{pk}$, and notice that σ_x consists of the $\omega_i = \mathsf{BV.Enc}_{\mathsf{pk}}(w_i)$, which are independent of f. Therefore, in the security experiment the challenger can answer the encoding queries before the adversary choses the function to attack on. At that point the challenger computes the remaining part of the public and secret key. Now, notice that the value c is statistically hidden to the server, because the entries of τ_i that depend on c (i.e. the first and the second entry) are masked by the corresponding uniform entry of ρ_i (resp. r_i, s_i). Now, for the sake of contradiction, suppose that a server could provide α', τ' with $\alpha' \neq \alpha$ such that the verification passes. This would imply that $\tau' = c \cdot \alpha' + \sum_{i=0}^{d} \rho_i \boxtimes \omega_i$, and that $\tau' - \tau = c \cdot (\alpha' - \alpha)$. Therefore, the server could compute c in polynomial time, by dividing (as an operation in R_q) the jth coordinate of $\tau' - \tau$ by the jth coordinate of $\alpha' - \alpha$. Though, computing c in polynomial time is impossible, as argued earlier, so a cheating server would succeed with negligible probability.

To show function privacy we argue that a simulator that does not know the function (or the secret key for the encryption scheme) can provide a transcript that is computationally close to the one provided by the honest client, therefore there is no information of the function revealed to the server. The simulator runs as follows: It runs **KeyGen** honestly, but for all i it sets $\alpha_i = \mathsf{BV.Enc}_{\mathsf{pk}}(0)$, instead. It runs **ProbGen** honestly, (which is possible since SK is independent of the function), and finally it runs **Verify** honestly (and does not need to perform the final decryption step). Since the encryption scheme is semantically secure, by a standard hybrid argument the server is provided with a transcript that is computationally indistinguishable from the honest one.

Input privacy can be argued in a similar way: it is guaranteed by the fact that the verification check is independent of the encoded values (therefore it does not reveal any information on the encrypted values) and all the other information about the secret weights given to the server is encrypted using a semantically secure encryption scheme, therefore a server cannot distinguish between a run with the real input and a run with dummy input.

In the above scheme we assume that the input to **ProbGen** is given in the so-called "streaming model", i.e. the w_i are generated in rounds and at each round **ProbGen** is called: this means that the vector σ_x is created at round 0 and sent to the server; then, at round i, it is updated by adding ω_i and ω_i is sent to the server. Analogously, τ_x is created and stored by the client at round 0 and at round i it is updated by adding $\rho_i \boxtimes \omega_i$. This allows the client to work with a short memory: indeed creating and sending the ω_i requires $O(\log(q))$ storage, as well as creating, updating and storing τ_x.

Moreover, as pointed out in the introduction, in case of a sparse linear combination, the client has two choices for the communication complexity: he can pay $O(\widetilde{d})$ by simply sending the non-zero weights encrypted (and their indices); however, he reveals the indices used in the linear combination (though not their values). If the client wants to completely hide the weights then it must pay $O(d)$ to send a ciphertext ω_i for each index, as in the standard scheme.

4.3 Polynomials

We now look at a more specific task, which is to perform the verifiable computation of functions for which $w_i = x^i$ for a value $x \in \mathbb{F}_p$ that does not need to be private.

Our solution is given by "blending" our scheme $\mathcal{VC}_{\mathsf{LC}}$ with the result in [BGV11], as follows: we modify the scheme in [BGV11] by assuming that the values t_i are authentications of the ciphertexts α_i, and that the verification check is carried out on the ciphertext α.

We use, again, the scheme in [BV11], but in this case we do not need its full power: we only make use of its additive homomorphic property, and apply the technique of [BGV11] on each component of the ciphertexts α_i, and verify the ciphertext output by the server component-wise. Again, acceptance or rejection by the client depends *only* on the correct execution by the server of the computation assigned to it over the ciphertexts. No useful decryption query is performed.

To be more formal, we assume that (as in [BGV11]) the client and the server agreed on a group G of prime order q in which the discrete logarithm problem is hard, and on a generator g for G.[5]

Now, as in the previous section, we use the [BV11] scheme, but in this case we only need its additively homomorphic property, so the ciphertext space is now $(R_q)^2$ (the last coordinate of the ciphertexts is superfluous), which, as an additive group is isomorphic to $(\mathbb{F}_q)^{2n}$. This observation is useful, because it allows to "encode" elements in the ambient space of the ciphertexts as discrete logarithms in G, as follows: for $\zeta \in (\mathbb{F}_q)^{2n}$, we define $g^\zeta := (g^{\zeta_1}, \ldots, g^{\zeta_{2^n}}) \in G^{2n}$. With this notation, we extend the operation of G coordinate-wise to G^{2n}. This is the main observation to implant the solution of [BGV11] into our scheme.

Before the specifications of the scheme, we shall make a final observation: due to the noise growth of the [BV11] scheme summarized in Theorem 1 (for $D = 1$), in order to achieve correctness of the result to be decoded by the client, we need q larger than $2 \cdot p^{d+1} \cdot \sigma \cdot n^{1.5}$ (the factor p^d on the right-hand side of the inequality is an upper bound on the norm of x^d).

The scheme $\mathcal{VC}_{\mathsf{poly}}$ is specified as follows:

1. **KeyGen**$(f = a_0, \ldots, a_d, \lambda) \to (PK, SK)$:
   - it runs $\mathsf{BV.ParamGen}(\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{BV.KeyGen}()$,
   - it specifies a group $(G, \cdot)$ of order q and a generator g,
   - it samples a uniform MAC key $c \leftarrow \mathbb{F}_q$, uniform $\kappa, \zeta \leftarrow (R_q)^2$,
   - it computes $\alpha_i \leftarrow \mathsf{BV.Enc}_{\mathsf{pk}}(a_i)$, $\tau_i \leftarrow c \cdot \alpha_i + \zeta^{*_{\mathbb{F}_q} i} *_{\mathbb{F}_q} \kappa$,[6] $G_{\tau,i} \leftarrow g^{\tau_i}$,
   - it sets $PK = (\mathsf{pk}, G, g, \alpha_0, G_{\tau,0}, \ldots, \alpha_d, G_{\tau,d})$ and $SK = (\mathsf{pk}, G, g, \mathsf{sk}, c, \zeta, \kappa)$.
2. **ProbGen**$_{SK}(x) \to \sigma_x, \tau_x$:
   - it sets $\sigma_x = x$, and $\tau_x = x$.
3. **Compute**$_{PK}(\sigma_x = x) \to \sigma_y$:
   - it computes $\alpha \leftarrow \sum_{i=0}^{d} x^i \cdot \alpha_i$, and $G_\tau \leftarrow \prod_{i=0}^{d} G_{\tau,i}^{x^i}$,
   - sets $\sigma_y = \alpha, G_\tau$

[5] In this section q is assumed to be a prime, while it was previously only assumed to be an integer co-prime to p.

[6] For any pair of vectors η, θ over $\mathbb{F}_q$, we denote by $\eta *_{\mathbb{F}_q} \theta$ the Schur (or Hadamard, or coordinate-wise) prouct of η with θ (over $\mathbb{F}_q$), that is the vector whose ith coordinate equals $\eta_i \cdot \theta_i$. Notice that $(R_q)^2$ is a vector space over $\mathbb{F}_q$ – indeed $(R_q)^2 \cong \mathbb{F}_q^{2n}$ – so the operation is well defined for elements in $(R_q)^2$.

4. **Verify**$_{SK}(\sigma_y = \alpha, G_\tau, \tau_x = x) \to (acc, a')$:
   - it computes the element μ in $(R_q)^2 \cong \mathbb{F}_q^{2\cdot n}$ such that for $j = 1, \ldots, 2 \cdot n$:
   $$\mu_j = \left((x \cdot \zeta_j)^{d+1} - 1\right) \cdot (x \cdot \zeta_j - 1)^{-1},$$
   - If $G_\tau \neq (g^\alpha)^c \cdot (g^\kappa)^{*_{\mathbb{F}_q}\mu}$, it rejects. Otherwise, it accepts and computes $a' \leftarrow \mathsf{BV.Dec_{sk}}(\alpha)$.

Theorem 3. *The scheme $\mathcal{VC}_{\mathsf{poly}}$ is correct, adaptively secure, and function private.*

Proof. For correctness, when both the client and the server are honest the verification step passes, because of the following reasoning. Notice that:

$$G_\tau = \prod_{i=0}^{d} G_{\tau,i}^{x^i} = \prod_{i=0}^{d} (g^{\tau_i})^{x^i} = g^{\sum_{i=0}^{d} \tau_i \cdot x^i} = g^{\sum_{i=0}^{d} \left(c\cdot\alpha_i + \zeta^{*_{\mathbb{F}_q} i} *_{\mathbb{F}_q} \kappa\right)\cdot x^i} =$$
$$= g^{c\cdot\sum_{i=0}^{d}\alpha_i\cdot x^i + \sum_{i=0}^{d}\zeta^{*_{\mathbb{F}_q} i} *_{\mathbb{F}_q} \kappa\cdot x^i} = g^{c\cdot\alpha} \cdot g^{\kappa *_{\mathbb{F}_q} \sum_{i=0}^{d}(\zeta\cdot x)^{*_{\mathbb{F}_q} i}}$$

Also for $j = 1, \ldots, 2 \cdot n$:

$$\left(\sum_{i=0}^{d} (\zeta \cdot x)^{*_{\mathbb{F}_q} i}\right)_j = \sum_{i=0}^{d} (\zeta_j \cdot x)^i = \sum_{i=0}^{d} \left((x \cdot \zeta_j)^{d+1} - 1\right) \cdot (x \cdot \zeta_j - 1)^{-1} = \sum_{i=0}^{d} \mu_j,$$

so $\sum_{i=0}^{d} (\zeta \cdot x)^{*_{\mathbb{F}_q} i} = \mu$. Therefore

$$g^{c\cdot\alpha} \cdot g^{\kappa *_{\mathbb{F}_q} \sum_{i=0}^{d}(\zeta\cdot x)^{*_{\mathbb{F}_q} i}} = g^{c\cdot\alpha} \cdot g^{\kappa *_{\mathbb{F}_q} \mu} = (g^\alpha)^c \cdot (g^\kappa)^{*_{\mathbb{F}_q}\mu}.$$

In the above notice that all the values written in Greek letters are elements in $(R_q)^2$ (therefore vectors over $\mathbb{F}_q$), while all the values in Latin letters are scalars in $\mathbb{F}_q$. This is important since all the sums and products at the exponent should be "compatible" with the group structure of G, which is of order q. In other words, the above formulas make sense because the additive group associated with each value at the exponent is $(\mathbb{F}_q, +)$, which is isomorphic to $(G, \cdot)$.
Moreover, the client retrieves the correct value of the computation, because

$$\mathsf{BV.Dec_{sk}}(\alpha) = \mathsf{BV.Dec_{sk}}\left(\sum_{i=0}^{d} \alpha_i \cdot x^i\right) = \mathsf{BV.Dec_{sk}}\left(\sum_{i=0}^{d} \mathsf{BV.Enc_{pk}}(a_i) \cdot x^i\right) = \sum_{i=0}^{d} a_i \cdot x^i.$$

Notice that the bound on x makes the last equality hold, since that bound respects the inequality in Theorem 1 (for $D = 1$).

For adaptive security, notice that $\sigma_x = x$, therefore the scheme is split. Moreover, its adaptive security property reduces to standard security, as the adversary has full knowledge on the encodings that can be queried in the adaptive game, even before seeing the public key of the verification scheme. Regular security can be proven in a game-based fashion, using a similar approach to the one in [BGV11], Section 5.2. We give a sketch here:

- Define Game_0 as the experiment in Definition 1.
- Define Game_1 as Game_1, but in which in the verification the value μ is computed via $\mu \leftarrow \sum_{i=0}^{d} (\zeta \cdot x)^{*_{\mathbb{F}_q} i}$.
- Define Game_2 as Game_1, but replacing the value $\zeta^{*_{\mathbb{F}_q} i} *_{\mathbb{F}_q} \kappa$ by a uniform $\nu_i \leftarrow (R_q)^2$.

As in [BGV11], the change between Game_1 and Game_0 is merely syntactical, so the advantage of a cheating server is the same. Moreover, the advantage of a cheating prover in Game_2 is negligibly close to the one in Game_1: we will show it by proving that the function $\varphi_{\kappa,\zeta} : i \mapsto g^{\zeta^{*_{\mathbb{F}_q} i} *_{\mathbb{F}_q} \kappa}$ is a pseudorandom function (based on the Strong Diffie-Hellmann assumption), for uniform $\kappa, \zeta \in (R_q)^2$. Notice first that $\varphi_{\kappa,\zeta}$ outputs vectors (over $\mathbb{F}_q$) whose entries are independently distributed, as $(\varphi_{\kappa,\zeta}(i))_j = g^{\zeta_j^i \cdot \kappa_j}$ depends only on the jth coordinate of ζ and κ, whose respective coordinates are sampled independently. Therefore, the pseudorandomness of $\varphi_{\kappa,\zeta}$ is granted by a standard hybrid argument, given that $(\varphi_{\kappa,\zeta}(\cdot))_j$ is an algebraic pseudorandom function, based on the Strong Diffie-Hellmann assumption.

Function privacy is guaranteed by a reduction to the semantic security of the scheme, similar to the reduction done in the proof of Theorem 2.

A possible application of the above protocol is to perform keyword search: specifically, let $F = (W_1, \ldots, W_d)$ be a file consisting of d words W_i. Let H be an injective function that maps "searchable" words W_j to integers of norm bounded by B. A client can encode F as the polynomial $\prod_{i=0}^{d}(X - H(W_i))$ and use its coefficients as the input to our protocol. Notice that this polynomial evaluates to zero only at those points that correspond (via H) to words in F, so in order to search for a word W the client can set $x \leftarrow H(W)$ and send it to the server which performs the evaluation of the polynomial. When the client decodes the answer y, after the verification, it interprets $y = 0$ as "W is in F" and $y \neq 0$ as "W is not in F".

Acknowledgments

The authors would like to thank Dario Fiore and Vinod Vaikuntanathan for useful discussions and suggestions.

The research of Rosario Gennaro was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

The research of Valerio Pastro was supported by NSF Grant No.1017660

References

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; a new characterization of np. In *FOCS*, pages 2–13. IEEE Computer Society, 1992.

[Bab85] László Babai. Trading group theory for randomness. In Robert Sedgewick, editor, *STOC*, pages 421–429. ACM, 1985.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[BCCT12a] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Goldwasser [Gol12], pages 326–349.

[BCCT12b] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. *IACR Cryptology ePrint Archive*, 2012. http://eprint.iacr.org/2012/095.

[BFL90] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. In *FOCS*, pages 16–25. IEEE Computer Society, 1990.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *STOC*, pages 21–31. ACM, 1991.

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Rogaway [Rog11], pages 111–131.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Goldwasser [Gol12], pages 309–325.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, pages 784–796, 2012.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Rogaway [Rog11], pages 505–524.

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

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

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[Gen10] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Rabin [Rab10], pages 116–137.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Rabin [Rab10], pages 465–482.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[GH11] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564, 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, *STOC*, pages 113–122. ACM, 2008.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier CS-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

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

[Gol12] Shafi Goldwasser, editor. *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*. ACM, 2012.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *STOC*, pages 723–732. ACM, 1992.

[Kil95] Joe Kilian. Improved efficient arguments (preliminary version). In Don Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 311–324. Springer, 1995.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 169–189. Springer, 2012.

[LPJY13] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly homomorphic structure-preserving signatures and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 289–307. Springer, 2013.

[Mic94] Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453. IEEE Computer Society, 1994.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012.

[Rab10] Tal Rabin, editor. *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010.

[RAD78] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. pages 169–177. Academic Press, 1978.

[Rog11] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013.