How to Delegate and Verify in Public: Verifiable Computation from Attribute-based Encryption

Bryan Parno[1], Mariana Raykova[⋆2], and Vinod Vaikuntanathan[⋆⋆3]

[1] Microsoft Research
[2] Columbia University
[3] University of Toronto

Abstract. The wide variety of small, computationally weak devices, and the growing number of computationally intensive tasks makes it appealing to delegate computation to data centers. However, outsourcing computation is useful only when the returned result can be trusted, which makes verifiable computation (VC) a must for such scenarios.

In this work we extend the definition of verifiable computation in two important directions: *public delegation* and *public verifiability*, which have important applications in many practical delegation scenarios. Yet, existing VC constructions based on standard cryptographic assumptions fail to achieve these properties.

As the primary contribution of our work, we establish an important (and somewhat surprising) connection between verifiable computation and attribute-based encryption (ABE), a primitive that has been widely studied. Namely, we show how to construct a VC scheme with public delegation and public verifiability from any ABE scheme. The VC scheme verifies any function in the class of functions covered by the permissible ABE policies (currently Boolean formulas). This scheme enjoys a very efficient verification algorithm that depends only on the output size. Efficient delegation, however, requires the ABE encryption algorithm to be cheaper than the original function computation. Strengthening this connection, we show a construction of a *multi-function* verifiable computation scheme from an ABE scheme with outsourced decryption, a primitive defined recently by Green, Hohenberger and Waters (USENIX Security 2011). A multi-function VC scheme allows the verifiable evaluation of multiple functions on *the same preprocessed input.*

In the other direction, we also explore the construction of an ABE scheme from verifiable computation protocols.

⋆ Research conducted as part of an internship with Microsoft Research.

⋆⋆ Supported by an NSERC Discovery Grant and by DARPA under Agreement number FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

1 Introduction

In the modern age of cloud computing and smartphones, asymmetry in computing power seems to be the norm. Computationally weak devices such as smartphones gather information, and when they need to store the voluminous data they collect or perform expensive computations on their data, they outsource the storage and computation to a large and powerful server (a "cloud", in modern parlance). Typically, the clients have a pay-per-use arrangement with the cloud, where the cloud charges the client proportional to the "effort" involved in the computation.

One of the main security issues that arises in this setting is – *how can the clients trust that the cloud performed the computation correctly?* After all, the cloud has the financial incentive to run (occasionally, perhaps) an extremely fast but incorrect computation, freeing up valuable compute time for other transactions. Is there a way to *verifiably outsource* computations, where the client can, without much computational effort, check the correctness of the results provided by the cloud? Furthermore, can this be done without requiring much interaction between the client and the cloud? This is the problem of *non-interactive verifiable computation*, which was considered implicitly in the early work on efficient arguments by Kilian [18] and computationally sound proofs (CS proofs) by Micali [20], and which has been the subject of much attention lately [2–5, 10, 11, 13, 14].

The starting point of this paper is that while the recent solutions consider and solve the bare-bones verifiable computation problem in its simplest form, there are a number of desirable features that they fail to achieve. We consider two such properties – namely, *public delegatability* and *public verifiability*.

Public Delegatability. In a nutshell, public delegatability says that everyone should be able to delegate computations to the cloud. In some protocols [2, 4, 10, 11], a client who wishes to delegate computation of a function F is required to first run an expensive pre-processing phase (wherein her computation is linear in the size of the circuit for F) to generate a (small) secret key SK_F and a (large) evaluation key EK_F. This large initial cost is then amortized over multiple executions of the protocol to compute $F(x_i)$ for different inputs x_i, but the client needs the secret key SK_F in order to initiate each such execution. In other words, *clients can delegate computation to the cloud only if they put in a large initial computational investment.* This makes sense only if the client wishes to run the same computation on many different inputs. Can clients delegate computation without making such a large initial commitment of resources?

As an example of a scenario where this might come in handy, consider a clinic with a doctor and a number of lab assistants, which wishes to delegate the computation of a certain expensive data analysis function F to a cloud service. Although the doctor determines the structure and specifics of F, it is in reality the lab assistants who come up with inputs to the function and perform the delegation. In this scenario, we would like to ask the doctor to run the (expensive) pre-processing phase once and for all, and generate a (small) *public*

key PK_F and an evaluation key EK_F. The public key lets anyone, including the lab assistants, delegate the computation of F to the cloud and verify the results. Thus, once the doctor makes the initial investment, any of the lab assistants can delegate computations to the cloud without the slightest involvement of the doctor. Needless to say, the cloud should not be able to cheat even given PK_F and EK_F.

Goldwasser, Kalai and Rothblum [13] present a *publicly delegatable* verifiable computation protocol for functions in the complexity class NC (namely, functions that can be computed by circuits of size $\mathsf{poly}(n)$ and depth $\mathsf{polylog}(n)$); indeed, their protocol is stronger in that it does not even require a pre-processing phase. In contrast, as mentioned above, many of the protocols for verifying general functions [2, 4, 10, 11] are *not publicly delegatable*. In concurrent work, Canetti, Riva, and Rothblum propose a similar notion (though they call it "public verifiability") [9] and construct a protocol, based on collision-resistant hashing and poly-logarithmic PIR, for general circuits C where the client runs in time $\mathtt{poly}(\log(|C|), \mathtt{depth}(C))$; they do not achieve the public verifiability property we define below. Computationally sound (CS) proofs achieve public delegatability; however the known constructions of CS proofs are either in the random oracle model [20], or rely on non-standard "knowledge of exponent"-type assumptions [5, 14]. Indeed, this seems to be an inherent limitation of solutions based on CS proofs since Gentry and Wichs [12] showed recently that CS proofs cannot be based on any falsifiable cryptographic assumption (using a black-box security reduction). Here, we are interested in standard model constructions, based on standard (falsifiable) cryptographic assumptions.

Public Verifiability. In a similar vein, the delegator should be able to produce a (public) "verification key" that enables anyone to check the cloud's work. In the context of the example above, when the lab assistants delegate a computation on input x, they can also produce a verification key VK_x that will let the patients, for example, obtain the answer from the cloud and check its correctness. Neither the lab assistants nor the doctor need to be involved in the verification process. Needless to say, the cloud cannot cheat even if it knows the verification key VK_x.

Papamanthou, Tamassia, and Triandopoulos [23] present a verifiable computation protocol for set operations that allows anyone who receives the result of the set operation to verify its correctness. In concurrent work, Papamanthou, Shi, and Tamassia [22] propose a similar notion, but they achieve it only for multivariate polynomial evaluation and differentiation, and the setup and evaluation run in time exponential in the degree; they do not consider the notion of public delegation. Neither the Goldwasser-Kalai-Rothblum protocol [13] nor any of the later works [2, 4, 10, 11] seem to be publicly verifiable.

Put together, we call a verifiable computation protocol that is both publicly delegatable and publicly verifiable a *public verifiable computation* protocol. We are not aware of any such protocol (for a general class of functions) that is non-interactive and secure in the standard model. Note that we still require the party who performs the initial function preprocessing (the doctor in the example above) to be trusted by those delegating inputs and verifying outputs.

As a bonus, a public verifiable computation protocol is immune to the "rejection problem" that affects several previous constructions [2, 10, 11]. Essentially, the problem is that these protocols do not provide reusable soundness; i.e., a malicious cloud that is able to observe the result of the verification procedure (namely, the accept/reject decision) on polynomially many inputs can eventually break the soundness of the protocol. It is an easy observation that public verifiable computation protocols do not suffer from the rejection problem. Roughly speaking, verification in such protocols depends only on the public key and some (instance-specific) randomness generated by the delegator, and not on any long-term secret state. Thus, obtaining the result of the verification procedure on one instance does not help break the soundness on a different instance.[1]

This paper is concerned with the design of public (non-interactive) verifiable computation protocols.

1.1 Our Results and Techniques

Our main result is a (somewhat surprising) connection between the notions of attribute-based encryption (ABE) and verifiable computation (VC). In a nutshell, we show that a *public* verifiable computation protocol for a class of functions $\mathcal{F}$ can be constructed from any attribute-based encryption scheme for a related class of functions – namely, $\mathcal{F} \cup \overline{\mathcal{F}}$. Recall that attribute-based encryption (ABE) is an rich class of encryption schemes where secret keys $\mathsf{ABE.SK}_F$ are associated with functions F, and can decrypt ciphertexts that encrypt a message m under an "attribute" x if and only if $F(x) = 1$.

For simplicity, we state all our results for the case of Boolean *functions*, namely functions with one-bit output. For functions with many output bits, we simply run independent copies of the verifiable computation protocol for each output bit.

Theorem 1 (Main Theorem, Informal). *Let $\mathcal{F}$ be a class of Boolean functions, and let $\overline{\mathcal{F}} = \{\overline{F} \mid F \in \mathcal{F}\}$ where $\bar{F}$ denotes the complement of the function F. If there is a key-policy ABE scheme for $\mathcal{F} \cup \overline{\mathcal{F}}$, then there is a public verifiable computation protocol for $\mathcal{F}$.*

Some remarks about this theorem are in order.

1. First, our construction is in the pre-processing model, where we aim to outsource the computation of the same function F on polynomially many inputs x_i with the goal of achieving an amortized notion of efficiency. This is the same as the notion considered in [10, 11], and different from the one in [13]. See Definition 1.
2. Secondly, since the motivation for verifiable computation is outsourcing computational effort, efficiency for the client is obviously a key concern. Our protocol will be efficient for the client, as long as computing an ABE encryption

[1] In fact, this observation applies also to any protocol that is publicly delegatable and not necessarily publicly verifiable.

(on input a message m and attribute x) takes less time than evaluating the function F on x. We will further address the efficiency issue in the context of concrete instantiations below (as well as in Section 3.2).

3. Third, we only need a *weak* form of security for attribute-based encryption which we will refer to as *one-key security*. Roughly speaking, this requires that an adversary, given a single key $\mathsf{ABE.SK}_F$ for any function F of its choice, cannot break the semantic security of a ciphertext under any attribute x such that $F(x) = 0$. Much research effort on ABE has been dedicated to achieving the much stronger form of security against collusion, namely when the adversary obtains secret keys for not just one function, but polynomially many functions of its choice. We will not require the strength of these results for our purposes. On the same note, constructing one-key secure ABE schemes is likely to be much easier than full-fledged ABE schemes.

A Note on Terminology: Attribute-based Encryption versus Predicate Encryption. In this paper, we consider attribute-based encryption (ABE) schemes to be ones in which each secret key $\mathsf{ABE.SK}_F$ is associated with a function F, and can decrypt ciphertexts that encrypt a message m under an "attribute" x if and only if $F(x) = 1$. This formulation is implicit in the early definitions of ABE introduced by Goyal, Pandey, Sahai and Waters [15, 25]. However, their work refers to F as an access structure, and existing ABE instantiations are restricted to functions (or access structures) that can be represented as polynomial-size span programs (a generalization of Boolean formulas) [15, 19, 21]. While such restrictions are not inherent in the definition of ABE, the fully general formulation we use above was first explicitly introduced by Katz, Sahai, and Waters, who dubbed it predicate encryption [17]. Note that we do not require attribute-hiding or policy/function-hiding, properties often associated with predicate encryption schemes (there appears to be some confusion in the literature as to whether attribute-hiding is inherent in the definition of predicate encryption [8, 17, 19], but the original formulation [17] does not seem to require it).

Thus, in a nutshell, our work can be seen as using ABE schemes for general functions, or equivalently, predicate encryption schemes that do not hide the attributes or policy, in order to construct verifiable computation protocols.

Let us now describe an outline of our construction. The core idea of our construction is simple: attribute-based encryption schemes naturally provide a way to "prove" that $F(x) = 1$. Say the server is given the secret key $\mathsf{ABE.SK}_F$ for a function F, and a ciphertext that encrypts a *random message* m under the attribute x. The server will succeed in decrypting the ciphertext and recovering m if and only if $F(x) = 1$. If $F(x) = 0$, he fares no better at finding the message than a random guess. The server can then prove that $F(x) = 1$ by returning the decrypted message.

More precisely, this gives an effective way for the server to convince the client that $F(x) = 1$. The pre-processing phase for the function F generates a master public key $\mathsf{ABE.MPK}$ for the ABE scheme (which acts as the public key for the verifiable computation protocol) and the secret key $\mathsf{ABE.SK}_F$ for the function F

(which acts as the evaluation key for the verifiable computation protocol). Given the public key and an input x, the delegator encrypts a random message m under the attribute x and sends it to the server. If $F(x) = 1$, the server manages to decrypt and return m, but otherwise, he returns $\perp$. Now,

- If the client gets back the same message that she encrypted, she is convinced beyond doubt that $F(x) = 1$. This is because, if $F(x)$ were 0, the server could not have found m (except with negligible probability, assuming the message is long enough).
- However, if she receives no answer from the server, it could have been because $F(x) = 0$ and the server is truly unable to decrypt, or because $F(x) = 1$ but the server intentionally refuses to decrypt.

Thus, we have a protocol with one-sided error – if $F(x) = 0$, the server can never cheat, but if $F(x) = 1$, he can.

A verifiable computation protocol with no error can be obtained from this by two independent repetitions of the above protocol – once for the function F and once for its complement $\bar{F}$. A verifiable computation protocol for functions with many output bits can be obtained by repeating the one-bit protocol above for each of the output bits. Intuitively, since the preprocessing phase does not create any secret state, the protocol provides public verifiable computation. Furthermore, the verifier performs as much computation as is required to compute two ABE encryptions.

Perspective: Signatures on Computation. Just as digital signatures authenticate messages, the server's proof in a non-interactive verifiable computation protocol can be viewed as a "signature on computation", namely a way to authenticate that the computation was performed correctly. Moni Naor has observed that identity-based encryption schemes give us digital signature schemes, rather directly [7]. Given our perspective, one way to view our result is as a logical extension of Naor's observation to say that just as IBE schemes give us digital signatures, ABE schemes give us signatures on computation or, in other words, non-interactive verifiable computation schemes.

Instantiations. Instantiating our protocol with existing ABE schemes creates challenges with regard to functionality, security, and efficiency. We discuss this issues briefly below and defer a detailed discussion to Section 3.2.

As mentioned earlier, existing ABE schemes only support span programs or polynomial-size Boolean formulas [15, 19, 21], which restricts us to this class of functions as well. In particular, the more recent ABE schemes, such as that of Ostrovsky, Sahai, and Waters [21], support the class of all (not necessarily monotone) formulas.

Another challenge is that most ABE schemes [15, 21, 25] are proven secure only in a selective-security model. As a result, instantiating the protocol above with such a scheme would inherit this limitation. If we instantiate our protocol with the scheme of Ostrovsky, Sahai, and Waters [21], we achieve a VC protocol for the class of polynomial-size Boolean formulas, which has delegation and verification algorithms whose combined complexity is more efficient than

the function evaluation. Essentially, the complexity gain arises because the delegation algorithm is essentially running the ABE encryption algorithm whose complexity is a fixed polynomial in $|x|$, the size of the input to the function, as well as the security parameter. The verification algorithm is very simple, involving just a one-way function computation. The resulting verifiable computation protocol is selectively secure.

Unfortunately, removing the "selective restriction" seems to be a challenge with existing ABE schemes. Although there have recently been constructions of adaptively secure ABE schemes, starting from the work of Lewko et al. [19], all these schemes work for *bounded* polynomial-size Boolean formulas. The up-shot is that the amount of work required to generate an encryption is proportional to the size of the formula, which makes the delegation as expensive as the function evaluation (and thus, completely useless)!

Much work in the ABE literature has been devoted to constructing ABE schemes that are secure against collusion. Namely, the requirement is that even if an adversary obtains secret keys for polynomially many functions, the scheme still retains security (in a precise sense). However, for our constructions, we require much less from the ABE scheme! In particular, we only need the scheme to be secure against adversaries that obtain the secret key for a single function. This points to instantiating our general construction with a *one-key secure* ABE scheme from the work of Sahai and Seyalioglu [24] for the class of bounded polynomial-size *circuits*. Unfortunately, because their scheme only supports bounded-size circuits, it suffers from the same limitation as that of Lewko et al. [19]. However, we can still use their construction to obtain a VC protocol where the *parallel complexity* of the verifier is significantly less than that required to compute the function.

We also note that when we instantiate our VC protocol with existing ABE schemes, the computation done by both the client and the worker is significantly cheaper than in any previous VC scheme, since we avoid the overhead of PCPs and FHE. However, existing ABE schemes restrict us to either formulas or a less attractive notion of parallel efficiency. It remains to be seen whether this efficiency can be retained while expanding the security offered and the class of functions supported. Fortunately, given the amount of interest in and effort devoted to new ABE schemes, we expect further improvements in both the efficiency and security of these schemes. Our result demonstrates that such improvements, as well as improvements in the classes of functions supported, will benefit verifiable computation as well.

1.2 Other Results

Multi-Function Verifiability and ABE with Outsourcing. The definition of verifiable computation focuses on the evaluation of a single function over multiple inputs. In many constructions [4, 10, 11] the evaluated function is embedded in the parameters for the VC scheme that are used for the input processing for the computation. Thus evaluations of multiple functions on the same input would require repeated invocation for the ProbGen algorithm. A notable difference are

approaches based on PCPs [5, 13, 14] that may require a single offline stage for input processing and then allow multiple function evaluations. However, such approaches inherently require verification work proportional to the depth of the circuit, which is at least logarithmic in the size of the function and for some functions can be also proportional to the size of the circuit. Further these approaches employ either fully homomorphic encryption or private information retrieval schemes to achieve their security properties.

Using the recently introduced definition of ABE with outsourcing [16], we achieve a multi-function verifiable computation scheme that decouples the evaluated function from the parameters of the scheme necessary for the input preparation. This VC scheme provides separate algorithms for input and function preparation, which subsequently can be combined for multiple evaluations. When instantiated with an existing ABE scheme with outsourcing [16], the verification algorithm for the scheme is very efficient: its complexity is linear in the output size but independent of the input length and the complexity of the computation. Multi-function VC provides significant efficiency improvements whenever multiple functions are evaluated on the same input, since a traditional VC scheme would need to invoke $\mathsf{ProbGen}$ for every function.

Attribute-Based Encryption from Verifiable Computation. We also consider the opposite direction of the ABE-VC relation: can we construct an ABE scheme from a VC scheme? We are able to show how to construct an ABE scheme from a *very special* class of VC schemes with a particular structure. Unfortunately, this does not seem to result in any new ABE constructions.

Due to space constraints, we defer the details to the full version of this paper.

2 Definitions

2.1 Public Verifiable Computation

We propose two new properties of verifiable computation schemes, namely

- *Public Delegation*, which allows arbitrary parties to submit inputs for delegation, and
- *Public Verifiability*, which allows arbitrary parties (and not just the delegator) to verify the correctness of the results returned by the worker.

Together, a verifiable computation protocol that satisfies both properties is called a *public verifiable computation* protocol. The following definition captures these two properties.

Definition 1 (Public Verifiable Computation). *A* public verifiable computation scheme (with preprocessing) $\mathcal{VC}$ *is a four-tuple of polynomial-time algorithms* $(\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *which work as follows:*

- $(PK_F, EK_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$*: The randomized* key generation *algorithm takes as input a security parameter* λ *and the function* F*, and outputs a public key* PK_F *and an evaluation key* EK_F*.*

– $(\sigma_x, VK_x) \leftarrow \mathsf{ProbGen}(PK_F, x)$*: The randomized* problem generation *algorithm uses the public key PK_F to encode an input x into public values σ_x and VK_x. The value σ_x is given to the worker to compute with, whereas VK_x is made public, and later used for verification.*

– $\sigma_{\mathsf{out}} \leftarrow \mathsf{Compute}(EK_F, \sigma_x)$*: The deterministic* worker *algorithm uses the evaluation key EK_F together with the value σ_x to compute a value σ_{out}.*

– $y \leftarrow \mathsf{Verify}(VK_x, \sigma_{\mathsf{out}})$*: The deterministic* verification *algorithm uses the verification key VK_x and the worker's output σ_{out} to compute a string $y \in \{0,1\}^* \cup \{\perp\}$. Here, the special symbol $\perp$ signifies that the verification algorithm rejects the worker's answer σ_{out}.*

A number of remarks on the definition are in order.

First, in some instantiations, the size of the public key (but not the evaluation key) will be independent of the function F, whereas in others, both the public key and the evaluation key will be as long as the description length of F. For full generality, we refrain from making the length of the public key a part of the syntactic requirement of a verifiable computation protocol, and instead rely on the definition of efficiency to enforce this (see Definition 4 below).

Secondly, our definition can be viewed as a "public-key version" of the earlier VC definition [10, 11]. In the earlier definition, KeyGen produces a secret key that was used as an input to ProbGen and, in turn, ProbGen produces a secret verification value needed for Verify (neither of these can be shared with the worker without losing security). Indeed, the "secret-key" nature of these definitions means that the schemes could be attacked given just oracle access to the verification function (and indeed, there are concrete attacks of this nature against the schemes in [2, 10, 11]). Our definition, in contrast, is stronger in that it allows any party holding the public key PK_F to delegate and verify computation of the function F on any input x, even if the party who originally ran ProbGen is no longer online. This, in turn, automatically protects against attacks that use the verification oracle.

Definition 2 (Correctness). *A verifiable computation protocol $\mathcal{VC}$ is* correct *for a class of functions $\mathcal{F}$ if for any $F \in \mathcal{F}$, any pair of keys $(PK_F, EK_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$, any $x \in \mathtt{Domain}(F)$, any $(\sigma_x, VK_x) \leftarrow \mathsf{ProbGen}(PK_F, x)$, and any $\sigma_{\mathsf{out}} \leftarrow \mathsf{Compute}(EK_F, \sigma_x)$, the verification algorithm* Verify *on input VK_x and σ_{out} outputs $y = F(x)$.*

Providing public delegation and verification introduces a new threat model in which the worker knows both the public key PK_F (which allows him to delegate computations) and the verification key VK_x for the challenge input x (which allows him to check whether his answers will pass the verification).

Definition 3 (Security). *Let $\mathcal{VC}$ be a public verifiable computation scheme for a class of functions $\mathcal{F}$, and let $A = (A_1, A_2)$ be any pair of probabilistic polynomial time machines. Consider the experiment $\mathbf{Exp}_A^{PubVerif}[\mathcal{VC}, F, \lambda]$ for any $F \in \mathcal{F}$ below:*

Experiment $\mathbf{Exp}_A^{PubVerif}[\mathcal{VC}, F, \lambda]$
$(PK_F, EK_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$;
$(x^*, \mathsf{state}) \leftarrow A_1(PK_F, EK_F)$;
$(\sigma_{x^*}, VK_{x^*}) \leftarrow \mathsf{ProbGen}(PK_F, x^*)$;
$\sigma^*_{\mathsf{out}} \leftarrow A_2(\mathsf{state}, \sigma_{x^*}, VK_{x^*})$;
$y^* \leftarrow \mathsf{Verify}(VK_{x^*}, \sigma^*_{\mathsf{out}})$
If $y^* \neq \perp$ *and* $y^* \neq F(x^*)$, *output '1', else output '0';*

A public verifiable computation scheme $\mathcal{VC}$ *is* secure *for a class of functions* $\mathcal{F}$, *if for every function* $F \in \mathcal{F}$ *and every p.p.t. adversary* $A = (A_1, A_2)$*:*

$$Pr[\mathbf{Exp}_A^{PubVerif}[\mathcal{VC}, F, \lambda] = 1] \leq \mathtt{negl}(\lambda). \tag{1}$$

where negl *denotes a negligible function of its input.*
Later, we will also briefly consider a weaker notion of "selective security" which requires the adversary to declare the challenge input x^* before it sees PK_F.

For verifiable outsourcing of a function to make sense, the client must use "less resources" than what is required to compute the function. "Resources" here could mean the running time, the randomness complexity, space, or the depth of the computation. We retain the earlier efficiency requirements [11] – namely, we require the complexity of ProbGen and Verify combined to be less than that of F. However, for KeyGen, we ask only that the complexity be $\mathsf{poly}(|F|)$. Thus, we employ an *amortized* complexity model, in which the client invests a larger amount of computational work in an "offline" phase in order to obtain efficiency during the "online" phase. We provide two strong definitions of efficiency – one that talks about the running time and a second that talks about computation depth.

Definition 4 (Efficiency). *A verifiable computation protocol* $\mathcal{VC}$ *is* efficient *for a class of functions* $\mathcal{F}$ *that act on* $n = n(\lambda)$ *bits if there is a polynomial* p *s.t.:* [2]

- *the running time of* ProbGen *and* Verify *together is at most* $p(n, \lambda)$*, the rest of the algorithms are probabilistic polynomial-time, and*
- *there exists a function* $F \in \mathcal{F}$ *whose running time is* $\omega(p(n, \lambda))$. [3]

In a similar vein, $\mathcal{VC}$ *is* depth-efficient *if the computation depth of* ProbGen *and* Verify *combined (written as Boolean circuits) is at most* $p(n, \lambda)$*, whereas there is a function* $F \in \mathcal{F}$ *whose computation depth is* $\omega(p(n, \lambda))$.

We now define the notion of unbounded circuit families which will be helpful in quantifying the efficiency of our verifiable computation protocols.

Definition 5. *We define a family of circuits* $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ *to be* unbounded *if for every polynomial* p *and all but finitely many* n*, there is a circuit* $C \in \mathcal{C}_n$ *of size at least* $p(n)$*. We call the family* depth-unbounded *if for every polynomial* p *and all but finitely many* n*, there is a circuit* $C \in \mathcal{C}_n$ *of depth at least* $p(n)$.

[2] To be completely precise, one has to talk about a family $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ parameterized by the input length n. We simply speak of $\mathcal{F}$ to implicitly mean $\mathcal{F}_n$ whenever there is no cause for confusion.

[3] This condition is to rule out trivial protocols, e.g., for a class of functions that can be computed in time less than $p(\lambda)$.

2.2 Key-Policy Attribute-Based Encryption

Introduced by Goyal, Pandey, Sahai and Waters [15], Key-Policy Attribute-Based Encryption (KP-ABE) is a special type of encryption scheme where a Boolean function F is associated with each user's key, and a set of attributes (denoted as a string $x \in \{0,1\}^n$) with each ciphertext. A key SK_F for a function F will decrypt a ciphertext corresponding to attributes x if and only if $F(x) = 1$. KP-ABE can be thought of as a special-case of predicate encryption [17] or functional encryption [8], although we note that a KP-ABE ciphertext need not hide the associated policy or attributes. We will refer to KP-ABE simply as ABE from now on. We state the formal definition below, adapted from [15, 19].

Definition 6 (Attribute-Based Encryption). *An attribute-based encryption scheme $\mathcal{ABE}$ for a class of functions $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ (where functions in $\mathcal{F}_n$ take n bits as input) is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *that work as follows:*

- $(PK, MSK) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ *: Given a security parameter λ and an index n for the family $\mathcal{F}_n$, output a public key PK and a master secret key MSK.*
- $C \leftarrow \mathsf{Enc}(PK, M, x)$*: Given a public key PK, a message M in the message space* MsgSp*, and attributes $x \in \{0,1\}^n$, output a ciphertext C.*
- $SK_F \leftarrow \mathsf{KeyGen}(MSK, F)$*: Given a function F and the master secret key MSK, output a decryption key SK_F associated with F.*
- $\mu \leftarrow \mathsf{Dec}(SK_F, C)$*: Given a ciphertext $C \in \mathsf{Enc}(PK, M, x)$ and a secret key SK_F for function F, output a message $\mu \in \mathsf{MsgSp}$ or $\mu = \perp$.*

Definition 7 (ABE Correctness). *Correctness of the ABE scheme requires that for all $(PK, MSK) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, all $M \in \mathsf{MsgSp}$, $x \in \{0,1\}^n$, all ciphertexts $C \leftarrow \mathsf{Enc}(PK, M, x)$ and all secret keys $SK_F \leftarrow \mathsf{KeyGen}(MSK, F)$, the decryption algorithm $\mathsf{Dec}(SK_F, C)$ outputs M if $F(x) = 1$ and $\perp$ if $F(x) = 0$. (This definition could be relaxed to hold with high probability over the keys (PK, MSK), which suffices for our purposes).*

We define a natural, yet relaxed, notion of security for ABE schemes which we refer to as "one-key security". Roughly speaking, we require that adversaries who obtain a single secret key SK_F for any function F of their choice and a ciphertext $C \leftarrow \mathsf{Enc}(PK, M, x)$ associated with any attributes x such that $F(x) = 0$ should not be able to violate the semantic security of C. We note that much work in the ABE literature has been devoted to achieving a strong form of security against collusion, where the adversary obtains not just a single secret key, but polynomially many of them for functions of its choice. We do not require such a strong notion for our purposes.

Definition 8 (One-Key Security for ABE). *Let $\mathcal{ABE}$ be a key-policy attribute-based encryption scheme for a class of functions $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$, and let $A = (A_0, A_1, A_2)$ be a three-tuple of probabilistic polynomial-time machines. We define security via the following experiment.*

Experiment $\mathbf{Exp}_A^{ABE}[\mathcal{ABE}, n, \lambda]$
$(PK, MSK) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$;
$(F, \mathsf{state}_1) \leftarrow A_0(PK)$;
$SK_F \leftarrow \mathsf{KeyGen}(MSK, F)$;
$(M_0, M_1, x^*, \mathsf{state}_2) \leftarrow A_1(\mathsf{state}_1, SK_F)$;
$b \leftarrow \{0,1\}$; $C \leftarrow \mathsf{Enc}(PK, M_b, x^*)$;
$\hat{b} \leftarrow A_2(\mathsf{state}_2, C)$;
If $b = \hat{b}$, *output '1', else '0';*

The experiment is valid if $M_0, M_1 \in \mathsf{MsgSp}$ *and* $|M_0| = |M_1|$. *We define the advantage of the adversary in all valid experiments as*

$$Adv_A(\mathcal{ABE}, n, \lambda) = |Pr[b = b'] - 1/2|.$$

We say that $\mathcal{ABE}$ *is a one-key secure ABE scheme if* $Adv_A(\mathcal{ABE}, n, \lambda) \leq \mathtt{negl}(\lambda)$.

3 Verifiable Computation from ABE

In Section 3.1, we present our main construction and proof, while Section 3.2 contains the various instantiations of our main construction and the concrete verifiable computation protocols that we obtain as a result.

3.1 Main Construction

Theorem 2. *Let* $\mathcal{F}$ *be a class of Boolean functions (implemented by a family of circuits* $\mathcal{C}$*), and let* $\overline{\mathcal{F}} = \{\overline{F} \mid F \in \mathcal{F}\}$ *where* $\bar{F}$ *denotes the complement of the function* F. *Let* $\mathcal{ABE}$ *be an attribute-based encryption scheme that is one-key secure (see Definition 8) for* $\mathcal{F} \cup \overline{\mathcal{F}}$, *and let* g *be any one-way function.*

Then, there is a verifiable computation protocol $\mathcal{VC}$ *(secure under Definition 3) for* $\mathcal{F}$. *If the circuit family* $\mathcal{C}$ *is unbounded (resp. depth-unbounded), then the protocol* $\mathcal{VC}$ *is efficient (resp. depth-efficient) in the sense of Definition 4.*

We first present our verifiable computation protocol.

Let $\mathcal{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$ be an attribute-based encryption scheme for the class of functions $\mathcal{F} \cup \overline{\mathcal{F}}$. Then, the verifiable computation protocol $\mathcal{VC} = (\mathsf{VC.KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ for $\mathcal{F}$ works as follows.[4] We assume, without loss of generality, that the message space $\mathcal{M}$ of the ABE scheme has size 2^λ.

Key Generation $\mathsf{VC.KeyGen}$**:** The client, on input a function $F \in \mathcal{F}$ with input length n, runs the ABE setup algorithm twice, to generate two independent key-pairs

$$(\mathsf{msk}_0, \mathsf{mpk}_0) \leftarrow \mathsf{ABE.Setup}(1^n, 1^\lambda) \quad \text{and} \quad (\mathsf{msk}_1, \mathsf{mpk}_1) \leftarrow \mathsf{ABE.Setup}(1^n, 1^\lambda)$$

[4] We denote the VC key generation algorithm as $\mathsf{VC.KeyGen}$ in order to avoid confusion with the ABE key generation algorithm.

Generate two secret keys $\mathsf{sk}_{\overline{F}} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{msk}_0, \overline{F})$ (corresponding to $\overline{F}$) and $\mathsf{sk}_F \leftarrow \mathsf{ABE.KeyGen}(\mathsf{msk}_1, F)$ (corresponding to F).

Output the pair $(\mathsf{sk}_{\overline{F}}, \mathsf{sk}_F)$ as the evaluation key and $(\mathsf{mpk}_0, \mathsf{mpk}_1)$ as the public key.

Delegation ProbGen**:** The client, on input x and the public key PK_F, samples two uniformly random messages $m_0, m_1 \xleftarrow{R} \mathcal{M}$, computes the ciphertexts

$$\mathsf{CT}_0 \leftarrow \mathsf{ABE.Enc}(\mathsf{mpk}_0, m_0) \qquad \text{and} \qquad \mathsf{CT}_1 \leftarrow \mathsf{ABE.Enc}(\mathsf{mpk}_1, m_1)$$

Output the message $\sigma_x = (\mathsf{CT}_0, \mathsf{CT}_1)$ (to be sent to the server), and the verification key $VK_x = (g(m_0), g(m_1))$, where g is the one-way function.

Computation Compute**:** The server, on receiving the ciphertexts $(\mathsf{CT}_0, \mathsf{CT}_1)$ and the evaluation key $EK_F = (\mathsf{sk}_{\overline{F}}, \mathsf{sk}_F)$ computes

$$\mu_0 \leftarrow \mathsf{ABE.Dec}(\mathsf{sk}_{\overline{F}}, \mathsf{CT}_0) \qquad \text{and} \qquad \mu_1 \leftarrow \mathsf{ABE.Dec}(\mathsf{sk}_F, \mathsf{CT}_1)$$

and send $\sigma_{\mathsf{out}} = (\mu_0, \mu_1)$ to the client.

Verification Verify**:** On receiving $VK_x = (v_0, v_1)$ and $\sigma_{\mathsf{out}} = (\mu_0, \mu_1)$, output [5]

$$y = \begin{cases} 0 \text{ if } g(\mu_0) = v_0 \text{ and } g(\mu_1) \neq v_1 \\ 1 \text{ if } g(\mu_1) = v_1 \text{ and } g(\mu_0) \neq v_0 \\ \perp \text{ otherwise} \end{cases}$$

Remark 1. Whereas our main construction requires only an ABE scheme, using an attribute-hiding ABE scheme (a notion often associated with predicate encryption schemes [8, 17]) would also give us input privacy, since we encode the function's input in the attribute corresponding to a ciphertext.

Remark 2. To obtain a VC protocol for functions with multi-bit output, we repeat this protocol (including the key generation algorithm) independently for every output bit. To achieve better efficiency, if the ABE scheme supports attribute hiding for a class of functions that includes message authentication codes (MAC), then we can define $F'(x) = MAC_K(F(x))$ and verify F' instead, similar to the constructions suggested by Applebaum, Ishai, and Kushilevitz [2], and Barbosa and Farshim [3].

Remark 3. The construction above requires the verifier to trust the party that ran ProbGen. This can be remedied by having ProbGen produce a non-interactive zero-knowledge proof of correctness [6] of the verification key VK_x. While theoretically efficient, the practicality of this approach depends on the particular ABE scheme and the NP language in question.

Proof of Correctness: The correctness of the VC scheme above follows from:

- If $F(x) = 0$, then $\overline{F}(x) = 1$ and thus, the algorithm Compute outputs $\mu_0 = m_0$ and $\mu_1 = \perp$. The algorithm Verify outputs $y = 0$ since $g(\mu_0) = g(m_0)$ but $g(\mu_1) = \perp \neq g(m_1)$, as expected.

[5] As a convention, we assume that $g(\perp) = \perp$.

– Similarly, if $F(x) = 1$, then $\overline{F}(x) = 0$ and thus, the algorithm Compute outputs $\mu_1 = m_1$ and $\mu_0 = \perp$. The algorithm Verify outputs $y = 1$ since $g(\mu_1) = g(m_1)$ but $g(\mu_0) = \perp \neq g(m_0)$, as expected.

■

We now consider the relation between the efficiency of the algorithms for the underlying ABE scheme and the efficiency for the resulting VC scheme. Since the algorithms Compute and Verify can potentially be executed by different parties, we consider their efficiency separately. It is easily seen that:

– The running time of the VC key generation algorithm VC.KeyGen is twice that of ABE.Setup plus ABE.KeyGen.
– The running time of Compute is twice that of ABE.Dec.
– The running time of ProbGen is twice that of ABE.Enc, and the running time of Verify is the same as that of computing the one-way function.

In short, the combined running times of ProbGen and Verify is polynomial in their input lengths, namely $p(n, \lambda)$, where p is a fixed polynomial, n is the length of the input to the functions, and λ is the security parameter. Assuming that $\mathcal{F}$ is an *unbounded* class of functions (according to Definition 5), it contains functions that take longer than $p(n, \lambda)$ to compute, and thus our VC scheme is efficient in the sense of Definition 4. (Similar considerations apply to depth-efficiency).

We now turn to showing the security of the VC scheme under Definition 3. We show that an attacker against the VC protocol must either break the security of the one-way function g or the one-key security of the ABE scheme.

Proof of Security: Let $A = (A_1, A_2)$ be an adversary against the VC scheme for a function $F \in \mathcal{F}$. We construct an adversary $B = (B_0, B_1, B_2)$ that breaks the one-key security of the ABE, working as follows. (For notational simplicity, given a function F, we let $F_0 = \overline{F}$, and $F_1 = F$.)

1. B_0 first tosses a coin to obtain a bit $b \in \{0, 1\}$. (Informally, the bit b corresponds to B's guess of whether the adversary A will cheat by producing an input x such that $F(x) = 1$ or $F(x) = 0$, respectively.)
   B_0 outputs the function F_b, as well as the bit b as part of the state.
2. B_1 obtains the master public key mpk of the ABE scheme and the secret key sk_{F_b} for the function F_b. Set $\mathsf{mpk}_b = \mathsf{mpk}$.
   Run the ABE setup and key generation algorithms to generate a master public key mpk' and a secret key $\mathsf{sk}_{F_{1-b}}$ for the function F_{1-b} under mpk'. Set $\mathsf{mpk}_{1-b} = \mathsf{mpk}'$.
   Let $(\mathsf{mpk}_0, \mathsf{mpk}_1)$ be the public key for the VC scheme and $(\mathsf{sk}_{F_0}, \mathsf{sk}_{F_1})$ be the evaluation key. Run the algorithm A_1 on input the public and evaluation keys and obtain a challenge input x^* as a result.
   If $F(x^*) = b$, output a uniformly random bit and stop. Otherwise, B_1 now chooses two uniformly random messages $M^{(b)}, \rho \leftarrow \mathcal{M}$ and outputs $(M^{(b)}, \rho, x^*)$ together with its internal state.
3. B_2 obtains a ciphertext $C^{(b)}$ (which is an encryption of either $M^{(b)}$ or ρ under the public key mpk_b and attribute x^*).

B_2 constructs an encryption $C^{(1-b)}$ of a uniformly random message $M^{(1-b)}$ under the public key mpk_{1-b} and attribute x^*.

Run A_2 on input $\sigma_{x^*} = (C^{(0)}, C^{(1)})$ and $VK_{x^*} = (g(M^{(0)}), g(M^{(1)}))$, where g is the one-way function. As a result, A_2 returns σ_{out}.

If $\mathsf{Verify}(VK_{x^*}, \sigma_{\mathsf{out}}) = b$, output 0 and stop.

We now claim the algorithms (B_0, B_1, B_2) described above distinguish between the encryption of $M^{(b)}$ and the encryption of ρ in the ABE security game with non-negligible advantage.

We consider two cases.

Case 1: $C^{(b)}$ is an encryption of $M^{(b)}$. In this case, B presents to A a perfect view of the execution of the VC protocol, meaning that A will cheat with probability $1/p(\lambda)$ for some polynomial p.

Cheating means one of two things. Either $F(x^*) = b$ and the adversary produced an inverse of $g(M^{(1-b)})$ (causing the Verify algorithm to output $1-b$), or $F(x^*) = 1-b$ and the adversary produced an inverse of $g(M^{(b)})$ (causing the Verify algorithm to output b).

In the former case, B outputs a uniformly random bit, and in the latter case, it outputs 0, the correct guess as to which message was encrypted. Thus, the overall probability that B outputs 0 is $1/2 + 1/p(\lambda)$.

Case 2: $C^{(b)}$ is an encryption of the message ρ. In this case, as above, B outputs a random bit if $F(x^*) = b$. Otherwise, the adversary A has to produce σ_{out} that makes the verifier output b, namely a string σ_{out} such that $g(\sigma_{\mathsf{out}}) = g(M^{(b)})$, while given only $g(M^{(b)})$ (and some other information that is independent of $M^{(b)}$).

This amounts to inverting the one-way function which A can only do with a negligible probability. (Formally, if the adversary wins in this game with non-negligible probability, then we can construct an inverter for the one-way function g).

The bottom line is that the adversary outputs 0 in this case with probability $1/2 + \mathtt{negl}(\lambda)$.

This shows that B breaks the one-key security of the ABE scheme with a non-negligible advantage $1/p(\lambda) - \mathtt{negl}(\lambda)$. ■

Remark 4. If we employ an ABE scheme that is selectively secure, then the construction and proof above still go through if we adopt a notion of "selectively-secure" verifiable computation in which the VC adversary commits in advance to the input on which he plans to cheat.

3.2 Instantiations

We describe two different instantiations of our main construction.

Efficient Selectively Secure VC Scheme for Formulas. The first instantiation uses the (selectively secure) ABE scheme of Ostrovsky, Sahai and Waters [21] for the

class of (not necessarily monotone) polynomial-size Boolean formulas (which itself is an adaptation of the scheme of Goyal et al. [15] which only supports monotone formulas[6]). This results in a selectively secure public VC scheme for the same class of functions, by invoking Theorem 2. Recall that selective security in the context of verifiable computation means that the adversary has to declare the input on which she cheats at the outset, before she sees the public key and the evaluation key.

The efficiency of the resulting VC scheme for Boolean formulas is as follows: for a boolean formula C, KeyGen runs in time $|C| \cdot \mathtt{poly}(\lambda)$; $\mathsf{ProbGen}$ runs in time $|x| \cdot \mathtt{poly}(\lambda)$, where $|x|$ is the length of the input to the formula; $\mathsf{Compute}$ runs in time $|C| \cdot \mathtt{poly}(\lambda)$; and Verify runs in time $O(\lambda)$. In other words, the total work for delegation and verification is $|x| \cdot \mathtt{poly}(\lambda)$ which is, in general, more efficient than the work required to evaluate the circuit C. Thus, the scheme is efficient in the sense of Definition 4. The drawback of this instantiation is that it is only selectively secure.

Recently, there have been constructions of fully secure ABE for formulas starting from the work of Lewko et al. [19] which, one might hope, leads to a fully secure VC scheme. Unfortunately, all known constructions of fully secure ABE work for *bounded* classes of functions. For example, in the construction of Lewko et al., once a bound B is fixed, one can design the parameters of the scheme so that it works for any formula of size at most B. Furthermore, implicit in the work of Sahai and Seyalioglu [24] is a construction of an (attribute-hiding, one-key secure) ABE scheme for *bounded* polynomial-size circuits (as opposed to formulas).

These constructions, unfortunately, do not give us efficient VC protocols. The reason is simply this: the encryption algorithm in these schemes run in time polynomial (certainly, at least linear) in B. Translated to a VC protocol using Theorem 2, this results in the worker running for time $\Omega(B)$ which is useless, since given that much time, he could have computed any circuit of size at most B by himself!

Essentially, the VC protocol that emerges from Theorem 2 is non-trivial if the encryption algorithm of the ABE scheme for the function family $\mathcal{F}$ is (in general) more efficient than computing functions in $\mathcal{F}$.

Depth-Efficient Adaptively Secure VC Scheme for Arbitrary Functions. Although the (attribute-hiding, one-key secure) ABE construction of Sahai and Seyalioglu [24] mentioned above does not give us an efficient VC scheme, it does result in a *depth-efficient VC scheme* for the class of polynomial-size circuits. Roughly speaking, the construction is based on Yao's Garbled Circuits, and involves an ABE encryption algorithm that constructs a garbled circuit for the function F

[6] Goyal et al.'s scheme [15] can also be made to work if we use DeMorgan's law to transform f and $\bar{f}$ into equivalent monotone formulas in which some variables may be negated. We then double the number of variables, so that for each variable v, we have one variable representing v and one representing its negation $\bar{v}$. Given an input x, we choose an attribute such that all of these variables are set correctly.

in question. Even though this computation takes at least as much time as computing the circuit for F, the key observation is that it can be done in parallel. In short, going through the VC construction in Theorem 2, one can see that both the Compute and Verify algorithms can be implemented in constant depth (for appropriate encryption schemes and one-way functions, e.g., the ones that result from the AIK transformation [1]), which is much faster in parallel than computing F, in general.

Interestingly, the VC protocol thus derived is very similar to the protocol of Applebaum, Ishai and Kushilevitz [2]. We refer the reader to [2, 24] for details.

We believe that this scheme also illuminates an interesting point: unlike other ABE schemes [15, 19, 21], this ABE scheme is only *one-key secure*, which is sufficient for verifiable computation. This relaxation may point the way towards an ABE-based VC construction that achieves generality, efficiency, and adaptive security.

4 Conclusions and Future Work

In this work, we introduced new notions for verifiable computation: *public delegatability* and *public verifiability*. We demonstrated a somewhat surprising construction of a public verifiable computation protocol from any (one-key secure) attribute-based encryption (ABE) scheme.

Our work leaves open several interesting problems. Perhaps the main open question is the design of *one-key secure* ABE schemes for general, unbounded classes of functions. Is it possible to come up with such a scheme for the class of all polynomial-size circuits (as opposed to circuits with an a-priori bound on the size, as in [24])? Given the enormous research effort in the ABE literature devoted to achieving the strong notion of security against collusion, our work points out that achieving even security against the compromise of a single key is a rather interesting question to investigate!

Acknowledgements. We wish to thank Seny Kamara and David Molnar for joint work in the early stages of this project, and Melissa Chase for her useful comments on this work. Our gratitude also goes to Daniel Wichs and the anonymous TCC reviewers whose comments helped improve the exposition of this paper.

References

1. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004.
2. B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2010.
3. M. Barbosa and P. Farshim. Delegatable homomorphic encryption with applications to secure outsourcing of computation. Cryptology ePrint Archive, Report 2011/215, 2011.

4. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *Proceedings of CRYPTO*, 2011.
5. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. Cryptology ePrint Archive, Report 2011/443, 2011.
6. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1988.
7. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
8. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *Proceedings of the Theory of Cryptography Conference (TCC)*, 2011.
9. R. Canetti, B. Riva, and G. N. Rothblum. Two 1-round protocols for delegation of computation. Cryptology ePrint Archive, Report 2011/518, 2011.
10. K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Proceedings of CRYPTO*, 2010.
11. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computation: Outsourcing computation to untrusted workers. In *Proceedings of CRYPTO*, 2010.
12. C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2011.
13. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the ACM Symposium on the Theory of Computing (STOC)*, 2008.
14. S. Goldwasser, H. Lin, and A. Rubinstein. Delegation of computation without rejection problem from designated verifier CS-proofs. Cryptology ePrint Archive, Report 2011/456, 2011.
15. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2006.
16. M. Green, S. Hohenberger, and B. Waters. Outsourcing the decryption of ABE ciphertexts. In *Proceedings of the USENIX Security Symposium*, 2011.
17. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of EuroCrypt*, 2008.
18. J. Kilian. Improved efficient arguments (preliminary version). In *Proceedings of CRYPTO*, 1995.
19. A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Proceedings of EuroCrypt*, 2010.
20. S. Micali. CS proofs (extended abstract). In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994.
21. R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2007.
22. C. Papamanthou, E. Shi, and R. Tamassia. Publicly verifiable delegation of computation. Cryptology ePrint Archive, Report 2011/587, 2011.
23. C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal verification of operations on dynamic sets. In *Proceedings of CRYPTO*, 2011.
24. A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2010.

25. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Proceedings of EuroCrypt*, 2005.