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

Bryan Parno
Microsoft Research
parno@microsoft.com

Mariana Raykova
Columbia University
mariana@cs.columbia.edu

Vinod Vaikuntanathan[*]
University of Toronto
vinodv@cs.toronto.edu

Abstract

The wide variety of small, computationally weak devices, and the growing number of computationally intensive tasks makes the delegation of computation to large data centers a desirable solution. However, computation outsourcing 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. This scheme enjoys a very efficient verification algorithm that depends only on the output size. Strengthening this connection, we show a construction of a *multi-function* verifiable computation scheme from an ABE 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.

[*]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 an extremely fast but incorrect computation (perhaps, once in a while), 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 [19] and computationally sound proofs (CS proofs) by Micali [22], and which has been the subject of much attention lately [1–4, 9, 11, 14, 15].

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. Everyone should be able to delegate computations to the cloud. In some of the protocols [1, 3, 9, 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 on different inputs x_i, and 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 having to make 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 that 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 computation of F to the cloud, and verify the results. Thus, once the doctor puts in 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 cannot cheat even if it knows the public key PK_F.

Goldwasser, Kalai and Rothblum [14] present a publicly delegatable verifiable computation protocol (In fact, their protocol does not require a pre-processing phase) for functions in the complexity class NC (namely, functions that can be computed by circuits of depth $\mathsf{polylog}(n)$). However, as mentioned above, the protocols in [1, 3, 9, 11] are *not publicly delegatable*. Computationally sound proofs achieve public delegatability, however the known constructions of CS proofs are either in the random oracle model [22], or rely on non-standard "knowledge of exponent" type assumptions [4, 15]. 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 be able to check the correctness of the answer. 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.

Neither the Goldwasser-Kalai-Rothblum protocol [14] nor any of the later works [1, 3, 9, 11] seem to be publicly verifiable. In fact, we are not aware of any non-interactive verifiable computation protocol (for a general class of functions) secure in the standard model which provides both public delegatability and public verifiability.

Put together, a verifiable computation protocol that is both publicly delegatable and publicly verifiable is called a *public verifiable computation* protocol. 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. In addition, those verifying results must trust the party (e.g., the lab assistant) that provided the verification key.

As a bonus, a public verifiable computation protocol is immune to the "rejection problem" that affects the constructions of [1, 9, 11]. A problem with the protocols of [1, 9, 11] is that they 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 – this is called the "rejection problem". It is an easy observation that public verifiable computation protocols do not suffer from the rejection problem. Roughly speaking, the reason is that 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.

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 particular, we show that any attribute-based encryption scheme for a class of functions $\mathcal{F}$ (that is closed under complement) can be used to construct a *public* verifiable computation protocol for $\mathcal{F}$.

Attribute based encryption schemes, a notion introduced by Goyal, Pandey, Sahai and Waters [16, 24], are a generalization of identity based encryption where secret keys $\mathsf{ABE.SK}_F$ associated to (Boolean) functions F can decrypt ciphertexts that encrypt a message m under "identity" x if and only if $F(x) = 1$. By now, there have been a number of constructions of ABE schemes for various classes of functions, the most general being an ABE scheme where F is any function that can be computed by a polynomial-size Boolean formula [16, 20].

Theorem 1 (Main Theorem, Informal) *Assume the existence one-way functions, and key-policy ABE schemes for a class of functions $\mathcal{F}$ closed under complement. Then, there is a public verifiable computation protocol for $\mathcal{F}$.*

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 identity 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 "identity" 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 repeating the above protocol twice, once for the function F and once for its complement $\overline{F}$. 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.

The key observation about this protocol is that the client's computation is polynomial in the length of its input x, the length of the public key PK_F and the security parameter – none of which depend on the complexity of the function F being evaluated. Thus, the client's computational complexity is independent of the complexity of the function F. Furthermore, with existing ABE schemes, the computation done by both the client and the worker is significantly cheaper than in any previous scheme, since we avoid the overhead of PCPs and FHE.

Thus far, most ABE schemes [16, 24] are proven secure only in a selective-security model. As a result, instantiating the protocol above with such a scheme would inherit this limitation. While it is believed that the ABE scheme of Goyal et al. [16] can be proven adaptively secure in the generic group model [26], the only scheme currently known to be adaptively secure is that of Lewko et al. [20]. Unfortunately, the protocol expands the attribute space such that there is essentially one attribute per *instance* of each variable in the Boolean formula. Thus, the amount of work required to generate an encryption is proportional to the size of the formula, making this scheme unattractive for outsourcing computation. Nonetheless, 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 will benefit verifiable computation as well.

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 [3, 9, 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 $\mathsf{ProbGen}$ algorithm. A notable difference are approached based on PCPs [4, 14, 15] 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 [17] we achieve 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. The verification algorithm for the scheme is very efficient and 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? In Appendix B, we show that we can indeed construct an ABE scheme from a VC scheme with a weak form of multi-function verifiability. Both VC schemes that we present in this paper can be modified so that they achieve the weak verifiability notion, and hence they can be used to instantiate the ABE construction.

1.2 Related Work

The goal of a VC scheme is to provide a way to efficiently verify work that has been outsourced to an untrusted party. Solutions for this problem have been proposed in various settings. These include interactive proofs [10,14,21,25] and interactive arguments [6,18,22]. However, in the context of delegated computation, a non-interactive approach for verifiability is much more desirable. CS proofs [22] realize a non-interactive argument in the random oracle model where the verification work is logarithmic in the complexity of the computation performed by the worker. Goldwasser, Kalai and Rothblum [14] construct a two message (non-interactive) protocol for functions in NC, where the verifier's running time depends on the depth of the circuit for the evaluated function.

The first solutions that provide verifiable computation schemes secure in the standard model for any polynomial-time computable function are the works of Gennaro, Gentry, and Parno [11] and Chung, Kalai, and Vadhan [9]. Both constructions employ fully homomorphic encryption for the evaluation of the delegated function, and neither can safely provide oracle access to the verification algorithm. This problem is resolved by Chung et al. [8], who consider the setting of memory delegation, where all inputs are preprocessed and given to the worker who will later execute multiple computations on them. Similar to the non-interactive solution of Goldwasser et al. [14], the effort required to verify results from memory delegation is proportional to the depth of the computation's circuit, which for certain functions may be proportional to the circuit size (e.g., exponentiation). The recent works of Bitansky et al. [4] and Goldwasser et al. [15] also achieve reusable soundness, though they rely on non-falsifiable "knowledge of exponent" type assumptions to so. Specifically, Bitansky et al. [4] present a construction for succinct non-interactive arguments based on a combination of PCP and PIR techniques, while Goldwasser et al. [15] give a construction for designated verifier CS proofs for polynomial functions, which also employs leveled fully homomorphic encryption.

Barbosa and Farshim [2] construct a verifiable computation protocol for arbitrary functions (without the rejection problem) from fully homomorphic encryption and functional encryption. Similar to the proposal of Applebaum, Ishai, and Kushilevitz [1], their protocol calculates a verifiable MAC over the computation's result, allowing efficient verification. However, this approach relies on powerful functional encryption functionality (e.g., the ability to compute MACs) that are currently not known to be achievable, whereas our approach needs only ABE.

The solutions of Benabbas, Gennaro, and Vahlis [3] and Papamanthou, Tamassia, and Triandopoulos [23] provide verifiable computation schemes for smaller classes of functions, polynomials and set operations respectively, but using more efficient tools than FHE or PCP combined with a single server PIR. Although VC schemes with reusable soundness protect against cheating even when the worker learns the output of the verification algorithm, they do not provide public verifiability where anyone can check the correctness of the result. The only exception is the work of Papamanthou et al. [23] which allows anyone who receives the result of the set operation to verify its correctness.

2 Definitions

We propose extended definitions for verifiable computation. We also summarize definitions for attribute-based encryption, since we explore its relationship with verifiable computation.

2.1 Public Verifiable Computation

Verifiable computation schemes enable a client to outsource the computation of a function F to an untrusted worker, and verify the correctness of the results returned by the worker [3, 9, 11, 14, 22]. Critically, the outsourcing and verification procedures must be significantly more efficient for the client than performing the computation by itself.

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 $\mathcal{VC}_{pub}$ *is a four-tuple of probabilistic polynomial-time algorithms* $(\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *which work as follows:*

- $\mathsf{KeyGen}(F, 1^\lambda) \rightarrow (PK_F, EK_F)$*: The randomized* key generation *algorithm takes as input a security parameter* λ *and the function* F*, and outputs a short public key PK that will be used for input delegation and a public evaluation key* EK_F*, which will be used for the evaluation of the function F (the length of this key will depend on the function).*
- $\mathsf{ProbGen}_{PK_F}(x) \rightarrow (\sigma_x, VK_x)$*: The* problem generation *algorithm uses the public key* PK_F *to encode the function input x as a public value* σ_x*, which is given to the worker to compute with, and a public value* VK_x*, which is used for verification.*
- $\mathsf{Compute}_{EK_F}(\sigma_x) \rightarrow \sigma_y$*: Using the public evaluation key and the encoded input, the worker* computes *an encoded version of the function's output* $y = F(x)$*.*
- $\mathsf{Verify}_{VK_x}(\sigma_y) \rightarrow y \cup \perp$*: Using the public verification key* VK_x*, the* public verification *algorithm converts the worker's output into the output of the function* $y = F(x)$*, or outputs* $\perp$ *indicating that* σ_y *does not represent the valid output of* F *on* x*.*

The changes relative to the original definition of verifiable computation [11] are with respect to KeyGen, ProbGen and Verify. In the original definition, KeyGen produced a secret key that was used as an input to ProbGen. ProbGen, in turn, produced a secret verification value needed for Verify; that verification value could not be shared with the worker. Indeed, previous schemes for general verifiable computation [9, 11] could be attacked given just oracle access to the verification function. This could be remedied by requiring the client to generate new parameters for the scheme, whenever the worker is caught cheating, but this is unattractive, given the computational overhead of generating parameters. A public verification key, in contrast, allows any party holding it to verify a computational result, even if the party that originally ran ProbGen is no longer online. Thus, this definition is strictly stronger than the earlier notion of security with verification access [11], which only gives the worker access to a verification oracle.

Providing public delegation and verifiability for outsourced computations introduces a new threat model in which the worker will be able to run the verification algorithm on any output that it intends to return. Accordingly, we modify the original security definition for verifiable computation [11] and require that even if the adversary can run ProbGen on his own, and he receives the verification key for the challenge input, he still cannot return an incorrect output that passes verification.

Definition 2 (Public Verifiable Computation Security) *Let $\mathcal{VC} = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a public verifiable computation scheme.*

```
Experiment $\mathbf{Exp}_A^{PubVerif}[\mathcal{VC}, F, \lambda]$
    $(PK_F, EK_F) \xleftarrow{R} \mathsf{KeyGen}(F, \lambda)$;
    $x \leftarrow A(PK_F, EK_F)$;
    $(\sigma_x, VK_x) \leftarrow \mathsf{ProbGen}_{PK_F}(x)$;
    $\hat{\sigma}_y \leftarrow A(PK_F, EK_F, \sigma_x, VK)$;
    $\hat{y} \leftarrow \mathsf{Verify}_{VK_x}(\hat{\sigma}_y)$
    If $\hat{y} \neq \perp$ and $\hat{y} \neq F(x)$, output '1', else '0';
```

In the experiment above, we define the advantage of an adversary A, running in probabilistic polynomial time and making a polynomial number of queries q, as:

$$Adv_A^{PubVerif}(\mathcal{VC}, F, \lambda, q) = Pr[\mathbf{Exp}_A^{PubVerif}[\mathcal{VC}, F, \lambda] = 1] \tag{1}$$

A public verifiable computation scheme $\mathcal{VC}$ is secure *for a function F, if*

$$Adv_A^{PubVerif}(\mathcal{VC}, F, \lambda, q) \leq \mathtt{negl}(\lambda). \tag{2}$$

where $\mathtt{negl}()$ is a negligible function of its input.

Efficiency requirements. For outsourcing work via verifiable computation to make sense, the client must to less work than that done by the worker. Thus, we retain the earlier efficiency requirements [11]; namely ProbGen and Verify must have smaller computational complexity than Compute. However, for KeyGen, we ask only that the complexity be $O(|F|)$; 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.

Remark 1 *Definition 2 is strictly stronger than the original definition of verifiable computation [11]. Hence any scheme satisfying Definition 2 will also satisfy the earlier definition.*

2.2 Multi-Function Verifiable Computation

The original definition of verifiable computation [11] assumed that multiple inputs would be prepared for a single function; here, we expand this definition to efficiently allow workers to verifiably apply multiple functions to a single input. In other words, previously, to evaluate $F(x)$ and $G(x)$, the client needed to run KeyGen for F, KeyGen for G, and then run ProbGen on x twice, once for F and once for G (since the public key PK used for the input preprocessing in ProbGen depends on the function that is evaluated). Our new definition only requires the client to run ProbGen once, and yet still allows the client to verify that a particular output was the output of a particular function on a particular input.

We present the multi-function property in the secret key setting of the original definition of verifiable computation [11], but note that it is orthogonal to the public delegation and verification defined in Section 2.1, and hence a scheme may have both properties, none, or one but not the other.

Since the original definition embeds the function to be computed in the scheme's parameters, we separate the generation of the parameters for the scheme, which will be used in ProbGen, into a Setup stage, and the generation of tokens for the evaluation of different functions into a KeyGen routine, which could be executed multiple times using the same parameters for the scheme. This allows the evaluation of multiple functions on the same instance produced by ProbGen.

Definition 3 (Multi-Function Verifiable Computation) *A VC scheme $\mathcal{VC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ is a multi-function verifiable computation scheme if it has the following properties:*

- $\mathsf{Setup}(\lambda) \to (PK_{param}, SK_{param})$: *Produces the public and private parameters that do not depend on the functions to be evaluated.*
- $\mathsf{KeyGen}_{PK_{param},SK_{param}}(F) \to (PK_F, SK_F)$: *Produces a keypair for evaluating and verifying a specific function F.*
- $\mathsf{ProbGen}_{PK_{param},SK_{param}}(x) \to (\sigma_x, \tau_x)$: *The algorithm requires the secret SK_{param}, which is independent of the function that will be computed. It generates both the encoding σ_x for the input, and the secret verification key τ_x.*
- $\mathsf{Compute}_{PK_{param},PK_F}(\sigma_x) \to \sigma_y$: *The computation algorithm uses both parts of the public key to produce an encoding of the output $y = F(x)$.*
- $\mathsf{Verify}_{SK_F,\tau_x}(\sigma_y) \to y \cup \perp$: *Using the private, function-specific key SK_F and the private, input-specific value τ_x, the verification algorithm converts the worker's output into $y = F(x)$, or outputs $\perp$ to indicate that σ_y does not represent a valid output of F on x.*

Definition 4 (Multi-Function Verifiable Computation Security) *Let $\mathcal{VC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a multi-function verifiable computation scheme. We define security via the following experiment.*

Experiment $\mathbf{Exp}_A^{MultVerif}[\mathcal{VC}, \lambda]$
$(PK_{param}, SK_{param}) \overset{R}{\leftarrow} \mathsf{KeyGen}(\lambda)$;
$(x, F, \hat{\sigma}_y) \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot),\mathcal{O}_{\mathsf{ProbGen}}(\cdot)}(PK_{param})$;
$\hat{y} \leftarrow \mathsf{Verify}_{SK_F,\tau_x}(\hat{\sigma}_y)$
If $\hat{y} \neq \perp$ *and* $\hat{y} \neq F(x)$, *output '1', else '0';*

We define the adversary's advantage and the scheme's security in the same fashion as Definition 2.

In the experiment, the adversary has oracle access to $\mathcal{O}_{\mathsf{KeyGen}}(F)$, which calls $\mathsf{KeyGen}_{PK_{param},SK_{param}}(F)$, returns PK_F, and stores SK_F. Similarly, the adversary can access the $\mathcal{O}_{\mathsf{ProbGen}}(\cdot)$ oracle, which calls $\mathsf{ProbGen}_{SK_{param}}(x)$, returns σ_x, and stores τ_x. Eventually, the adversary returns an encoding $\hat{\sigma}_y$ which purports to be an output of F applied to x. The challenger runs Verify with the corresponding values of τ_x and SK_F, and the adversary wins if this check passes.

2.3 Key-Policy Attribute-Based Encryption

Introduced by Goyal et al. [16], Key-Policy Attribute-Based Encryption (KP-ABE) can be thought of as associating a function F with each user's key, and a set of attributes with each ciphertext. A key will decrypt a particular ciphertext only if the key's function evaluates to true for the attributes associated with the ciphertext. In this sense, KP-ABE can be thought of as a special-case of Functional Encryption [5].

2.3.1 Basic KP-ABE

Currently known KP-ABE constructions support functions that are poly-sized Boolean formulas ϕ in n variables; ciphertexts are associated with attributes of the form $\vec{z} = (z_1, \ldots, z_n) \in \{0,1\}^n$. However, this restricted set of functions is not inherent in the definition of KP-ABE. Thus, we state the following definitions, adapted from the definitions of Goyal et al. [16] and Lewko et al. [20], in terms of a general policy function.

Definition 5 (Key-Policy Attribute-Based Encryption) *An attribute-based encryption scheme $\mathcal{ABE}$ is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ defined as follows:*

- $\mathsf{Setup}(\lambda, U) \to (PK, MSK)$: *Given a security parameter λ and the set of all possible attributes U, output a public key PK and a master secret key MSK.*
- $\mathsf{Enc}_{PK}(M, \gamma) \to C$: *Given a public key PK, a message M, and a set of attributes γ, output ciphertext C.*
- $\mathsf{KeyGen}_{MSK}(F) \to SK_F$: *Given a function F and the master secret key MSK, output a decryption key SK_F associated with that function.*

- $\mathsf{Dec}_{SK_F}(C) \to M \cup \perp$: *Given a ciphertext $C = \mathsf{Enc}_{PK}(M,\gamma)$ and a secret key SK_F for function F, output M if $F(\gamma) = 1$, or $\perp$, otherwise.*

Below, we give the natural definition for KP-ABE security. Early KP-ABE schemes were proven secure in the weaker "selective-security" model, but the stronger definition below was recently achieved by Lewko et al. [20].

Definition 6 (KP-ABE Security) *Let $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ be a key-policy attribute-based encryption scheme. We define security via the following experiment.*

Experiment $\mathbf{Exp}_A^{ABE}[\mathcal{ABE}, F, U, \lambda]$
$(PK, MSK) \overset{R}{\leftarrow} \mathsf{Setup}(\lambda, U)$;
$(M_0, M_1, \hat{\gamma}) \leftarrow A^{O_{\mathsf{KeyGen}}(\cdot)}(PK)$;
$b \leftarrow \{0,1\}$;
$\hat{b} \leftarrow A^{O_{\mathsf{KeyGen}}(\cdot)}(PK, \mathsf{Enc}_{PK}(M_b))$;
If $b = \hat{b}$, output '1', else '0';

In the experiment, the adversary has access to an oracle $O_{\mathsf{KeyGen}}(F)$, which invokes $SK_F \leftarrow \mathsf{KeyGen}_{MSK}(F)$ and returns SK_F. Eventually, $\mathcal{A}$ chooses two messages M_0, M_1 of equal length and a set of challenge attributes γ, and he receives the encryption of one of two messages. Ultimately, he must decide which of the two plaintext messages was encrypted.

We consider the experiment valid *if $\forall SK_F \in \mathcal{R} : F(\gamma) \neq 1$, where $\mathcal{R} = \{SK_F\}$ is the set of responses to the oracle. In other words, the adversary cannot hold a key that trivially decrypts messages encrypted under the challenge attribute γ.*

We define the advantage of the adversary in all valid experiments as

$$Adv_{\mathcal{A}}(\mathcal{ABE}, U, \lambda) = \left| Pr[b = b'] - \frac{1}{2} \right|.$$

We say that $\mathcal{ABE}$ is a secure key-policy attribute-based encryption scheme if $Adv_{\mathcal{A}}(\mathcal{ABE}, U, \lambda) < negl(\lambda)$.

2.3.2 KP-ABE With Outsourcing

Green, Hohenberger, and Waters define a notion of ABE with outsourcing in which the party performing the decryption can offload most of the decryption work to an untrusted third party [17]; the third party learns nothing about the underlying plaintext, and the party holding the secret key can complete the decryption very efficiently, in time independent of the size of the formula associated with the key. Although they define and construct ABE with outsourcing for both CP-ABE and KP-ABE, below, we focus on the definitions for KP-ABE, which will be relevant for our work. We also give an IND-CPA security definition, since we do not require the stronger RCCA they defined. Note that Green et al.'s construction [17] is selectively secure, but they provide a sketch, based on Lewko et al.'s work [20], to show that their scheme can also be made adaptively secure.

Note that in this context the outsourcing done is for the very specific function of ABE partial decryption. The definitions also do not include a notion of integrity or verification, as in verifiable computation, but instead are concerned with the secrecy of the underlying plaintext.

Definition 7 (Key-Policy Attribute-Based Encryption With Outsourcing [17]) *A KP-ABE scheme with outsourcing, $\mathcal{ABE}$, is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Transform}, \mathsf{Dec})$ defined as follows:*

- $\mathsf{Setup}(\lambda, U) \to (PK, MSK)$: *Given a security parameter λ and the set of all possible attributes U, output a public key PK and a master secret key MSK.*
- $\mathsf{Enc}_{PK}(M, \gamma) \to C$: *Given a public key PK, a message M, and a set of attributes γ, output ciphertext C.*

- $\mathsf{KeyGen}_{MSK}(F) \to (SK_F, TK_F)$: *Given a function F and the master secret key MSK, output a decryption key SK_F and a transformation key TK_F associated with that function.*
- $\mathsf{Transform}_{TK_F}(C) \to C' \cup \perp$: *Given a ciphertext $C = \mathsf{Enc}_{PK}(M, \gamma)$ and a transformation key TK_F for function F, output a partially decrypted ciphertext C' if $F(\gamma) = 1$, or $\perp$, otherwise.*
- $\mathsf{Dec}_{SK_F}(C') \to M \cup \perp$: *Given a partially decrypted ciphertext $C' = \mathsf{Transform}_{TK_F}(\mathsf{Enc}_{PK}(M, \gamma))$ and a secret key SK_F for function F, output M if $F(\gamma) = 1$, or $\perp$, otherwise.*

Definition 8 (KP-ABE With Outsourcing IND-CPA Security) *Let $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Transform}, \mathsf{Dec})$ be a key-policy attribute-based encryption scheme with outsourcing. We define security via the following experiment.*

```
Experiment $\mathbf{Exp}_A^{ABE-Out}[\mathcal{ABE}, F, U, \lambda]$
    $(PK, MSK) \overset{R}{\leftarrow} \mathsf{Setup}(\lambda, U)$;
    $(M_0, M_1, \gamma) \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot), \mathcal{O}_{\mathbf{Corrupt}}(\cdot)}(PK)$;
    $b \leftarrow \{0, 1\}$;
    $\hat{b} \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot), \mathcal{O}_{\mathbf{Corrupt}}(\cdot)}(PK, \mathsf{Enc}_{PK}(M_b, \gamma))$ ;
    If $b = \hat{b}$, output '1', else '0';
```

In the experiment, the adversary has access to two oracles. $\mathcal{O}_{\mathsf{KeyGen}}(F)$ invokes $(SK_F, TK_F) \leftarrow \mathsf{KeyGen}_{MSK}(F)$, stores SK_F and returns TK_F. $\mathcal{O}_{\mathbf{Corrupt}}(F)$ returns SK_F if the adversary previously invoked $\mathcal{O}_{\mathsf{KeyGen}}(F)$ and returns $\perp$ otherwise. Eventually, $\mathcal{A}$ chooses two messages M_0, M_1 of equal length and a set of challenge attributes γ, and he receives the encryption of one of two messages. Ultimately, he must decide which of the two plaintext messages was encrypted.

We consider the experiment valid *if $\forall SK_F \in \mathcal{R} : F(\gamma) \neq 1$, where $\mathcal{R} = \{SK_F\}$ is the set of valid responses to the $\mathcal{O}_{\mathbf{Corrupt}}(F)$ oracle. In other words, the adversary cannot hold a key that trivially decrypts messages encrypted under the challenge attribute γ.*

We define the advantage of the adversary in all valid experiments as

$$Adv_{\mathcal{A}}(\mathcal{ABE}, U, \lambda) = \left| Pr[b = b'] - \frac{1}{2} \right|.$$

We say that $\mathcal{ABE}$ is a secure key-policy attribute-based encryption scheme with outsourcing if $Adv_{\mathcal{A}}(\mathcal{ABE}, U, \lambda) < negl(\lambda)$.

3 Public Verifiable Computation from Attribute-Based Encryption

The security definition of KP-ABE guarantees that decryption will only succeed when the function associated with a secret key evaluates to a particular check value (traditionally "1") on the attribute associated with the ciphertext. Seen in this light, we can view the decrypted plaintext value from a KP-ABE ciphertext as an efficiently verifiable proof that the result of applying the key's function to the attribute value is equal exactly to the check value.

Thus, in the verifiable computation setting, if we give the worker a key associated with the function we wish to evaluate, then we can encrypt random messages, using as attributes the input values for the computation. The worker's response will simply be the decryptions that he manages to obtain. The only valid plaintext values that he can return will be associated with attributes that satisfy the check value for the function evaluation. We can make this scheme publicly verifiable by defining the verification tokens to be the result of applying a one-way function to the plaintexts. Indeed this transformation applies to any VC scheme in which verification consists of a simple comparison operation.

One problem with this approach is that the worker may still cheat by returning nothing; i.e., claiming that none of the ciphertexts decrypted successfully, even when some did succeed. To address this problem, consider a function $f : X \to \{0,1\}$ with binary output. We can compute the complement function $\overline{f}$, which always outputs the opposite bit of the output of f. If we give the worker secret keys for *both f and $\overline{f}$*, and encrypt two *different* random messages, then the worker must return exactly one of the messages, since exactly one of the ciphertexts will always decrypt successfully. To keep the ciphertexts distinct, we generate the keys for f and $\overline{f}$ (and the corresponding encryptions) using two distinct master keypairs for the KP-ABE system.

To extend this idea to functions with multi-bit outputs, we decompose each function F into subfunctions $f_1, \ldots, f_n$ where f_i represents the function that computes the i^{th} output bit of F. Alternately, if the KP-ABE scheme supports attribute hiding (also known as predicate encryption) and a class of functions that includes MACs, then we can define $F'(x) = MAC_K(F(x))$ and verify F' instead, similar to the constructions suggested by Applebaum, Ishai, and Kushilevitz [1], and Barbosa and Farshim [2].

An attribute hiding ABE or predicate encryption scheme would also give us input and output privacy, without the need for fully-homomorphic encryption. Attribute hiding naturally gives us input privacy, since we encode the function's input in the attribute. By randomly permuting the keypairs and ciphertexts we give out, we can also hide whether a successful decryption corresponds to a 0 or 1 output.

We present a construction of a public verifiable computation scheme for functions with one bit output from a KP-ABE scheme follow. The extension to multi-bit outputs is straightforward as described above.

Construction 1 ($\mathcal{VC}_{ABE}$) *Let $\mathcal{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Dec})$ be a KP-ABE scheme for a class of functions $\mathcal{F}$ that is closed under complement with possible attributes from the set U, and let g be a one-way function.*

Then there is a public verifiable computation scheme $\mathcal{VC} = (\mathsf{VCKeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ for the same class of functions $\mathcal{F}$, which is defined as follows.

- $\mathsf{VCKeyGen}(f^0, \lambda) \to (PK_F, EK_F)$, *where $f^0 \in \mathcal{F}$ is such that $f^0 : \{0,1\}^n \to \{0,1\}$:*
  1. *Generate two key pairs by running*
  $$(MPK^0_{abe}, MSK^0_{abe}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^n) \quad \text{and} \quad (MPK^1_{abe}, MSK^1_{abe}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, U)$$
  2. *Set*
  $$SK^{f^0}_{abe} = \mathsf{ABE.KeyGen}_{MSK^0_{abe}}(f^0) \quad \text{and} \quad SK^{f^1}_{abe} = \mathsf{ABE.KeyGen}_{MSK^1_{abe}}(f^1)$$
  where f^1 denotes the complement of the function f^0, i.e., $f^1(x) = 1$ if and only if $f^0(x) = 0$.
  3. *Let the public key for the verifiable computation be $PK_F = (MPK^0_{abe}, MPK^1_{abe})$ and the evaluation key be $EK_F = (SK^{f^0}_{abe}, SK^{f^1}_{abe})$.*
- $\mathsf{ProbGen}_{PK_F}(x) \to (\sigma_x, VK_x)$*:*
  1. *Generate a pair of random messages $(m_0, m_1) \xleftarrow{R} \{0,1\}^\lambda \times \{0,1\}^\lambda$.*
  2. *Compute ciphertexts $c_0 = \mathcal{ABE}.\mathsf{Enc}_{MPK^0_{abe}}(m_0, x)$ and $c_1 = \mathcal{ABE}.\mathsf{Enc}_{MPK^1_{abe}}(m_1, x)$.*
  3. *Output (σ_x, VK_x), where $\sigma_x = (c_0, c_1)$ and $VK_x = (g(m_0), g(m_1))$.*
- $\mathsf{Compute}_{EK_F}(\sigma_x) \to \sigma_y$ *:*
  1. *Parse EK_F as $(SK^{f^0}_{abe}, SK^{f^1}_{abe})$ and σ_x as (c_0, c_1).*
  2. *Compute $d_0 = \mathcal{ABE}.\mathsf{Dec}_{SK^{f^0}_{abe}}(c_0)$ and $d_1 = \mathcal{ABE}.\mathsf{Dec}_{SK^{f^1}_{abe}}(c_1)$.*
  3. *Output $\sigma_y = (d_0, d_1)$.*
- $\mathsf{Verify}_{VK_x}(\sigma_y) \to y \cup \perp$*:*
  1. *Parse VK_x as $(g(m_0), g(m_1))$ and σ_y as (d_0, d_1).*
  2. *If $g(d_0) = g(m_0)$ then $y = 0$, else if $g(d_1) = g(m_1)$, then $y = 1$, else output $\perp$.*

Efficiency. To be a true verifiable computation scheme, $\mathcal{VC}_{ABE}$ must satisfy the efficiency requirements from Section 2.1; specifically, the running time of $\mathsf{ProbGen}$ and Verify must be less than computing the function itself. Since Verify simply requires two one-way function computations and two equality checks, the runtime is independent of the function. For any "reasonable" KP-ABE scheme, the runtime of $\mathsf{ProbGen}$ should be independent of F as well, since all of the inputs to Enc are independent of F.

To prove the security of $\mathcal{VC}_{ABE}$, we show that a VC attacker must break the security of the one-way function g or of the KP-ABE scheme. Intuitively, if an adversary can break $\mathcal{VC}_{ABE}$, then he will return the decryption of a message encrypted with an attribute that does not satisfy the function being verified. This decryption can then be used to decide which plaintext was encrypted in the KP-ABE challenge.

Theorem 2 *Let $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ be a secure (according to Definition 6) key-policy attribute-based encryption scheme for a class of function $\mathcal{F}$. Let $\mathcal{VC}_{ABE} = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be the verifiable computation scheme obtained from $\mathcal{ABE}$ using Construction 1. Then $\mathcal{VC}_{ABE}$ is secure according to Definition 2.*

The proof of Theorem 2 is given in Appendix A.

Remark 2 *If we employ a KP-ABE scheme that is selectively secure, then the construction and proof above still go through, as long as the KP-ABE scheme's attribute domain has size polynomial in the security parameter. In that case, $\mathcal{A}_{ABE}$ simply guesses the attribute the VC adversary will cheat on. Since $\mathcal{A}_{VC}$ cheats with non-negligible probability, and the probability that he will cheat at x is $\frac{1}{poly(\lambda)}$, it follows that $\mathcal{A}_{ABE}$ will also have non-negligible advantage in the ABE security game.*

Remark 3 *If we employ a KP-ABE scheme that is selectively secure with a larger domain, 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.*

4 Multi-Function Verifiable Computation from KP-ABE With Outsourcing

The original definition of KP-ABE does not readily lend itself to multi-function verifiable computation. Specifically, it does not allow the client an easy way to verify which function was used to compute an answer. For example, suppose the client gives out keys SK_F and SK_G for functions F and G. Following Construction 1, to outsource computation on input x, the client gives out (among other things) a ciphertext $\mathsf{Enc}_{PK}(M_0, x)$. Now, suppose $F(x) = 1$, but $G(x) \neq 1$. The worker can use SK_F to obtain M_0, but claim that this output corresponds to a computation of G. In essence, Construction 1 gives us a way to verify that an output corresponds to a particular input, but if we give out more than one secret key, it cannot distinguish between functions. One remedy would be to run two parallel instances of Construction 1, but then we need to run $\mathsf{ProbGen}$ for each function we wish to compute on a given input.

A more elegant solution is to use an ABE scheme that requires an extra step to decrypt a ciphertext. Thus, we show how to build multi-function verifiable computation from KP-ABE with outsourcing [17] (see Section 2.3.2). We use the transformation key to allow the worker to compute, and then use the secret key as a verification key for the function. This allows us to verify both the input and specific function used to compute a particular result returned by the worker.

Interestingly, a similar scheme can be constructed from Chase's multi-authority ABE [7], by using function identifiers (e.g., a hash of the function description, or a unique ID assigned by the client) in place of user identifiers, and using the "user key" generated by the Central Authority as a verification token for a particular function. However, since this approach does not employ the multi-authority ABE scheme in a black-box fashion, in this section, we focus on the construction from KP-ABE with outsourcing.

We specify the construction in detail below. For clarity, we only consider functions with single-bit outputs, but the construction can be generalized just as we did in Section 3.

Construction 2 *Let* $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{TransformDec})$ *be a KP-ABE scheme with outsourcing with attribute universe* U. *We construct a multi-function verifiable computation scheme as follows:*

- $\mathsf{Setup}(\lambda) \rightarrow (PK_{param}, SK_{param})$ *: Run* $\mathcal{ABE}.\mathsf{Setup}(\lambda, U)$ *twice to obtain* (PK^0, MSK^0) *and* (PK^1, MSK^1). *Set* $PK_{param} = (PK^0, PK^1)$ *and* $SK_{param} = (MSK^0, MSK^1)$.
- $\mathsf{KeyGen}_{PK_{param}, SK_{param}}(F) \rightarrow (PK_F, SK_F)$ *: Compute* $(SK_F^0, TK_F^0) \leftarrow \mathcal{ABE}.\mathsf{KeyGen}_{MSK^0}(F)$ *and* $(SK_{\overline{F}}^1, TK_{\overline{F}}^1) \leftarrow \mathcal{ABE}.\mathsf{KeyGen}_{MSK^1}(\overline{F})$, *where* $\overline{F}$ *is the complement of* F.
  Output $PK_F = (TK_F^0, TK_{\overline{F}}^1)$ *and* $SK_F = (SK_F^0, SK_{\overline{F}}^1)$. *In other words, the public key will be the transformation keys, and the secret verification key will be the "true" secret keys.*
- $\mathsf{ProbGen}_{PK_{param}, SK_{param}}(x) \rightarrow (\sigma_x, \tau_x)$*: Generate a pair of random messages* $(M_0, M_1) \overset{R}{\leftarrow} \{0,1\}^\lambda \times \{0,1\}^\lambda$. *Compute ciphertexts* $C_0 \leftarrow \mathcal{ABE}.\mathsf{Enc}_{PK_0}(M_0, x)$ *and* $C_1 \leftarrow \mathcal{ABE}.\mathsf{Enc}_{PK_1}(M_1, x)$.
  Output $\sigma_x = (C_0, C_1)$ *and* $\tau_x = (M_0, M_1)$.
- $\mathsf{Compute}_{PK_{param}, PK_F}(\sigma_x) \rightarrow \sigma_y$*: Parse* PK_F *as* $(TK_F^0, TK_{\overline{F}}^1)$. *Compute* $C_0' = \mathcal{ABE}.\mathsf{Transform}_{TK_F^0}(C_0)$ *and* $C_1' = \mathcal{ABE}.\mathsf{Transform}_{TK_{\overline{F}}^1}(C_1)$. *Output* $\sigma_y = (C_0', C_1')$.
- $\mathsf{Verify}_{SK_F, \tau_x}(\sigma_y) \rightarrow y$*: Parse* SK_F *as* $(SK_F^0, SK_{\overline{F}}^1)$, τ_x *as* (M_0, M_1), *and* σ_y *as* (C_0', C_1'). *If* $\mathcal{ABE}.\mathsf{Dec}_{SK_F^0}(C_0') = M_0$, *then output* $y = 0$. *If* $\mathcal{ABE}.\mathsf{Dec}_{SK_{\overline{F}}^1}(C_0') = M_1$, *then output* $y = 1$. *Otherwise, output* $\perp$.

The above construction will provide the efficiency property required for a VC scheme (verification that is more efficient than the delegated computation) as long as the Transform algorithm is computationally more expensive than the Encand Decalgorithms of the ABE scheme. However, this requirement is inherent in the definition of ABE with outsourcing.

Remark 4 *Construction 2 is publicly delegatable, since* ProbGen *only makes use of* PK_{param}*; i.e., it only employs the public ABE keys to perform* ProbGen, *so anyone may do so. However, the verification function cannot be made public. Specifically, giving out* SK_F *in Green et al.'s ABE scheme [17] would directly allow the worker to lie about the function used, i.e., claim that an output computed with* F *was the result of applying* G. *Even so, the adversary would still be unable to lie about the output's* value.

Proof Intuition. The proof of security looks very similar to Proof A. The intuition for input security is the same as before, i.e., the revelation of one of the two random messages associated with a ProbGen invocation demonstrates that the computation was performed on that particular input. Unlike with a regular ABE scheme, we can also verify the function used, since decrypting with a key that does not match the transformation key used will not produce the expected message. This is provided by the security of the ABE with outsourced decryption, which guarantees semantic security of the encrypted messages even when the adversary sees the transformation keys used for the outsourced portion of the decryption.

5 Conclusions and Future Work

In this work, we introduced new notions for verifiable computation: *public delegation*, *public verifiability*, and *multi-function* VC. We demonstrate that ABE implies public VC, and ABE with outsourcing implies multi-function VC. The relationship offers much more efficient VC schemes for the class of functions supported by ABE.

Our work leaves open several interesting problems. First, can we achieve public verifiability for a multi-function VC scheme? Second, it would be desirable to show a MAC scheme that can be supported by an ABE scheme's class of functions, so that we can improve the efficiency of verifying multi-bit outputs.

References

[1] 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.

[2] M. Barbosa and P. Farshim. Delegatable homomorphic encryption with applications to secure outsourcing of computation. Cryptology ePrint Archive, Report 2011/215, 2011.

[3] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *Proceedings of CRYPTO*, 2011.

[4] 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.

[5] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *Proceedings of the IACR Theory of Cryptography Conference (TCC)*, 2011.

[6] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2), 1988.

[7] M. Chase. Multi-authority attribute based encryption. In *Proceedings of the IACR Theory of Cryptography Conference (TCC)*, 2007.

[8] K.-M. Chung, Y. Kalai, F.-H. Liu, and R. Raz. Memory delegation. In *Proceedings of CRYPTO*, 2011.

[9] K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Proceedings of CRYPTO*, Aug. 2010.

[10] L. Fortnow and C. Lund. Interactive proof systems and alternating time-space complexity. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS)*, 1991.

[11] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computation: Outsourcing computation to untrusted workers. In *Proceedings of CRYPTO*, Aug. 2010.

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

[13] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1989.

[14] 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.

[15] 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.

[16] 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.

[17] M. Green, S. Hohenberger, and B. Waters. Outsourcing the decryption of ABE ciphertexts. In *Proceedings of the USENIX Security Symposium*, 2011.

[18] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1992.

[19] J. Kilian. Improved efficient arguments (preliminary version). In *Proceedings of CRYPTO*, 1995.

[20] 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.

[21] C. Lund, L. Fortnow, and H. Karloff. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

[22] S. Micali. CS proofs (extended abstract). In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1994.

[23] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal verification of operations on dynamic sets. In *Proceedings of CRYPTO*, 2011.

[24] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[25] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[26] B. Waters. Private communications, 2011.

A Proof of Security for the ABE to VC Construction

Proof of Theorem 2:

Suppose there exists an adversary $\mathcal{A}_{VC}$ who succeeds in the VC security game (Definition 2) with non-negligible probability when the VC scheme is instantiated with the ABE-based scheme from Construction 1. We first argue that if the ABE scheme is secure, then $\mathcal{A}_{VC}$ cannot distinguish between the following two games:

Game 0. We run the real security game for the VC construction.

Game 1. We run the real security game for the VC scheme with the following modification. When the adversary supplies us with the challenge input x, instead of returning an encryption of m_0 and an encryption of m_1, we choose a random message $m' \neq m_0, m_1$, let r be such that $f^r(x) = 0$, and replace c_r with $\mathcal{ABE}.\mathsf{Enc}_{PK^r_{abe}}(m', x)$.

If $\mathcal{A}_{VC}$ succeeds in distinguishing between these two games with non-negligible probability, i.e., if

$$\left|Pr\left[\mathbf{Exp}^0_A[\mathcal{VC}, F, \lambda]\right] - Pr\left[\mathbf{Exp}^1_A[\mathcal{VC}, F, \lambda]\right]\right| > \delta$$

where δ is non-negligible in λ, then we can construct an adversary $\mathcal{A}_{ABE}$ against the ABE scheme as follows.

1. $\mathcal{A}_{ABE}$ interacts with the challenger $\mathcal{C}$ in the ABE security game and acts as the challenger for $\mathcal{A}_{VC}$ in the security game for the verifiable computation.
2. $\mathcal{A}_{ABE}$ receives a public key PK^0_{abe} from $\mathcal{C}$, and runs $\mathcal{ABE}.\mathsf{Setup}(\lambda, U)$ on its own to generate a second pair of keys $(PK^1_{abe}, MSK^1_{abe})$.
3. Let f^0 be the function we wish to verify. $\mathcal{A}_{ABE}$ computes $f^1 \leftarrow \overline{f^0}$, the complement function of f^0.
4. $\mathcal{A}_{ABE}$ chooses a random bit $r \xleftarrow{R} \{0,1\}$ and generates $SK^{f^r}_{abe} = \mathcal{ABE}.\mathsf{KeyGen}_{MSK^1_{abe}}(f^r)$ using his own master secret key. He then makes a call to $\mathcal{C}$ to obtain $SK^{f^{1-r}}_{abe} = \mathcal{ABE}.\mathsf{KeyGen}_{MSK^0_{abe}}(f^{1-r})$. $\mathcal{A}_{ABE}$ sends $PK_F = \{PK^0_{abe}, PK^1_{abe}\}$ and $EK_F = \{SK^{f^{1-r}}_{abe}, SK^{f^r}_{abe}\}$ to $\mathcal{A}_{VC}$.
5. Upon receiving the challenge input x, $\mathcal{A}_{ABE}$ chooses three random messages m_0, m_1 and m_2. $\mathcal{A}_{ABE}$ sends m_0, m_1, and attribute $\gamma = x$ to its challenger and receives challenge ciphertext c, which is the encryption of m_b. $\mathcal{A}_{ABE}$ sets $\sigma_x \leftarrow (c, \mathcal{ABE}.\mathsf{Enc}_{PK^1_{abe}}(m_2, \gamma))$, chooses a random $t \xleftarrow{R} \{0,1\}$, sets $VK_x \leftarrow (g(m_t), g(m_2))$, and sends (σ_x, VK_x) to $\mathcal{A}_{VC}$.
6. $\mathcal{A}_{ABE}$ receives the result $\hat{\sigma}_y$ from $\mathcal{A}_{VC}$.
7. If $g(\hat{\sigma}_y) = g(m_t)$, then $\mathcal{A}_{ABE}$ guesses $b' \leftarrow t$. Otherwise, $\mathcal{A}_{ABE}$ guesses $b' \leftarrow 1 - t$.

We observe that if $t = b$, the distribution of the above experiment coincides with Game 0. If $t = 1 - b$, the distribution is the same as Game 1. Thus, we have:

$$\begin{aligned}
Pr[b' = b] &= Pr[t = b]Pr[b' = b|t = b] + Pr[t \neq b]Pr[b' = b|t \neq b] \\
&= \frac{1}{2}Pr[g(\hat{\sigma}_y) = g(m_t)|t = b] + \frac{1}{2}Pr[g(\hat{\sigma}_y) \neq g(m_t)|t \neq b] \\
&= \frac{1}{2}\mathbf{Exp}_A^0[\mathcal{VC}, F, \lambda] + \frac{1}{2}(1 - Pr[g(\hat{\sigma}_y) = g(m_t)|t \neq b]) \\
&= \frac{1}{2}\mathbf{Exp}_A^0[\mathcal{VC}, F, \lambda] + \frac{1}{2}(1 - \mathbf{Exp}_A^1[\mathcal{VC}, F, \lambda]) \\
&= \frac{1}{2}(\mathbf{Exp}_A^0[\mathcal{VC}, F, \lambda] - \mathbf{Exp}_A^1[\mathcal{VC}, F, \lambda] + 1) \\
&= \frac{1}{2}(\mathbf{Exp}_A^0[\mathcal{VC}, F, \lambda] - \mathbf{Exp}_A^1[\mathcal{VC}, F, \lambda] + 1) \\
&\geq \frac{1}{2}(\delta + 1)
\end{aligned}$$

which implies

$$Adv_{\mathcal{A}_{ABE}} = \left| Pr[b = b'] - \frac{1}{2} \right| \geq \left| \frac{1}{2}(\delta + 1) - \frac{1}{2} \right| = \frac{\delta}{2}$$

Thus, if $\mathcal{A}_{VC}$ distinguishes between these two games, then $\mathcal{A}_{ABE}$ can distinguish which ciphertext his challenger encrypted with non-negligible probability.

Now, since $\mathcal{A}_{VC}$ cannot distinguish between Game 0 and Game 1, we can run him in Game 1, and use him to invert the one-way function g. Specifically, suppose we are given $z = g(w)$, and we want to find w. We run Game 1 with $\mathcal{A}_{VC}$ using ABE keys we have generated, and we define $m_r = w$. Thus, we will give $\mathcal{A}_V$ the value z as the verification token for output r. Since $f^{1-r}(x) = 1$, a legitimate worker would return m_{1-r}, so to cheat in the VC experiment, the $\mathcal{A}_{VC}$ must return m_r such that $g(m_r) = z$. In other words, the value σ_y he returns must be w, and hence we will invert the one-way function with non-negligible probability.

Thus, assuming the ABE scheme and the one-way function are secure, Construction 1 is also secure.

■

B Attribute-based Encryption from Verifiable Computation

Given that we have shown how to construct a verifiable computation (VC) protocol from an attribute-based encryption (ABE) scheme, it is natural to ask whether the reverse implication holds. In other words, can we construct an ABE scheme, given a VC scheme? At first sight, the key property of a VC scheme – namely, efficient verification – does not seem to have anything to do with attribute-based encryption.

Despite this apparent mismatch of functionality, we show how to transform (publicly) verifiable computation protocols of a special form – that we call weak "multi-function verifiable computation" – into an attribute-based encryption scheme for the same class of functions. Informally, a weak multi-function VC protocol has the following features:

- The output of ProbGen on an input x can be used to compute many different functions on x. Thus, in some sense, ProbGen is agnostic of the function that will be computed on the input.

  In particular, we now have a setup algorithm that generates a pair of public and secret parameters, a key generation algorithm KeyGen (as before) that generates a secret key SK_F for a function F given the secret parameters, and a ProbGen algorithm that (as before) given an input x and the public parameters generates an encoding of x together with a verification key. Thus, $\mathsf{ProbGen}$ does not know about F and KeyGen does not know about x. Indeed, this is the crucial property that gives us ABE.

- The verification key for an input x consists of a pair (VK_x^0, VK_x^1), and the verification algorithm consists of simply applying some function H to the server's response σ_y and checking if it equals VK_x^0 or VK_x^1.

Indeed, the VC scheme that we constructed from ABE has both the above properties.

The high level idea for the construction of an ABE scheme from a multi-function VC scheme is as follows: in order to encrypt a message under a particular attribute (in the ABE scheme), we first generate a key that can be computed only if the output of the server in the VC protocol verifies correctly. Now, decryption of the ciphertext will succeed only if the decryptor correctly performs the evaluation of the key's function on the attribute associated with the ciphertext, and the output value of the computation satisfies the decryption condition, in which case he will have the correct decryption key for the ciphertext.

Put another way, the security of a VC scheme implies it should be difficult for an adversary to produce an output that does not correspond to a legitimate computation of a function on a particular input. If we make decryption of a ciphertext dependent on having a particular output, then the computation possible given a key for the function and an attribute/input either legitimately produces the expected output, allowing decryption of the ciphertext, or produces some other output, and it is infeasible to produce the output necessary to decrypt the ciphertext.

We define the notion of a weak multi-function VC protocol below. The "weakness" in the definition comes from the fact that we only need a multi-function VC scheme that verifies that a particular output is legitimate for *some* outsourced function (i.e., a function given as input to KeyGen), rather than for a specific function.

Below, we combine these requirements in a single definition, demonstrate that both Constructions 1 and 2 satisfy this definition, and finally show how to use such a definition to construct an ABE scheme.

Definition 9 (Weak Multi-Function Public Verifiable Computation) *A VC scheme $\mathcal{VC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ is a weak multi-function public verifiable computation scheme if it has the following properties:*

- $\mathsf{Setup}(\lambda) \to (PK_{param}, SK_{param})$*: Produces the public and private parameters that do not depend on the functions to be evaluated.*
- $\mathsf{KeyGen}_{PK_{param}, SK_{param}}(F) \to PK_F$*: Produces a public key for evaluating a specific function F.*
- $\mathsf{ProbGen}_{PK_{param}}(x) \to (\sigma_x, VK_x = (VK_x^0, VK_x^1))$*: The algorithm requires only the public parameters, which are independent of the function that will be computed. It generates both the encoding σ_x for the input, and the public verification keys for each possible bit of the output, in this case, simply VK_x^0 and VK_x^1.*
- $\mathsf{Compute}_{PK_{param}, PK_F}(\sigma_x) \to \sigma_y$*: The computation algorithm uses both parts of the public key to produce an encoding of the output $y = F(x)$.*
- $\mathsf{Verify}_{VK_x}(\sigma_y) \to y \cup \perp$*: Using the public input-specific value VK_x, the verification algorithm outputs 0 if $VK_x^0 = H(\sigma_y)$, outputs 1 if $VK_x^1 = H(\sigma_y)$, and outputs $\perp$ otherwise, to indicate that σ_y does not represent a valid output of some function F, for which $\mathsf{KeyGen}(F)$ has been invoked, on x*

Definition 10 (Weak Multi-Function Public Verifiable Computation Security) *Let $\mathcal{VC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a weak multi-function public verifiable computation scheme. We define security via the following experiment.*

```
Experiment $\mathbf{Exp}_A^{WeakMultVerif}[\mathcal{VC},\lambda]$
    $(PK_{param}, SK_{param}) \xleftarrow{R} \mathsf{KeyGen}(\lambda)$;
    $x \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot)}(PK_{param})$;
    $(\sigma_x, VK_x) \leftarrow \mathsf{ProbGen}_{PK_{param}}(x)$;
    $\hat{\sigma}_y \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot)}(PK_{param}, \sigma_x, VK_x)$;
    $\hat{y} \leftarrow \mathsf{Verify}_{VK_x}(\hat{\sigma}_y)$
    If $\hat{y} \neq \perp$ and $\forall F \in \mathcal{R} : \hat{y} \neq F(x)$, output '1', else '0';
```

We define the adversary's advantage and the scheme's security in the same fashion as Definition 2.

In the experiment, the adversary has oracle access to $\mathcal{O}_{\mathsf{KeyGen}}(F)$, which calls $\mathsf{KeyGen}_{PK_{param},SK_{param}}(F)$, returns PK_F, and stores F in the list $\mathcal{R}$. Eventually, the adversary returns an encoding $\hat{\sigma}_y$ which purports to be an output of some outsourced function applied to x. The challenger runs Verify with the corresponding values of VK_x, and the adversary wins if this check passes, but the output does not correspond to the output of one of the functions in list $\mathcal{R}$.

Note that both Constructions 1 and 2 satisfy this definition. Construction 1 does not include any function verification to begin with, but it still verifies that the output, i.e., the message obtained after performing a decryption, could not have been obtained without performing a legitimate computation (decryption) with one of the keys generated by KeyGen. In contrast, Construction 2 is too strong, since it verifies the specific function used. To weaken it, we can simply add the private decryption key SK_F to the computation key, which was previously TK_F. This removes the ability to verify which function was used, and hence fits within the definition above.

Below, we describe our construction from VC to ABE in more detail.

Construction 3 *Let $\mathcal{VC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a weak multi-function public verifiable computation scheme, and H be an injective one way function. We construct the following key-policy attribute-based encryption scheme $\mathcal{ABE}_{VC}$.*

- $\mathsf{Setup}(\lambda, U) \rightarrow (PK, MSK)$: *Run $\mathcal{VC}.\mathsf{Setup}(\lambda) \rightarrow (PK_{param}, SK_{param})$ and output $PK = PK_{param}$ and $MSK = SK_{param}$.*
- $\mathsf{Enc}_{PK}(M, \gamma) \rightarrow C$ *where $M \in \{0,1\}$:*
  - *Run $(\sigma_x, VK_x = (VK_x^0, VK_x^1)) \leftarrow \mathcal{VC}.\mathsf{ProbGen}_{PK_{params}}(x)$.*
  - *Let σ_{ans} be such that $H(\sigma_{ans}) = VK_x^1$. Choose a random value r and compute $K = \langle \sigma_{ans}, r \rangle$ where $\langle \cdot, \cdot \rangle$ denotes the inner product of two bit-strings (mod 2).*

  Output ciphertext $C = (\sigma_x, r, K \oplus M)$.
- $\mathsf{KeyGen}_{MSK}(F) \rightarrow SK_F$: *Run $PK_F \leftarrow \mathcal{VC}.\mathsf{KeyGen}_{PK_{param},SK_{param}}(F)$. Output $SK_F = PK_F$.*
- $\mathsf{Dec}_{SK_F}(C) \rightarrow M \cup \perp$: *Parse C as (σ_x, r, D).*
  - *Run $\sigma_{ans} \leftarrow \mathcal{VC}.\mathsf{Compute}_{PK_{params},PK_F}(\sigma_x)$.*
  - *Compute $K \leftarrow \langle \sigma_{ans}, r \rangle$. Output $K \oplus D$.*

Correctness follows from the fact that if $F(x) = 1$, then the answer σ_{ans} produced by the server (upon running $\mathcal{VC}.\mathsf{Compute}$) is such that $H(\sigma_{ans}) = VK_x^1$. Since H is an injective one-way function, $\langle \sigma_{ans}, r \rangle \oplus D = M$.

We now proceed to showing the security of the ABE scheme. First, we state the Goldreich-Levin lemma [13].

Lemma 1 ([13]) *Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ be a bijection computable by a circuit of size t and suppose there is a circuit $\mathcal{C}$ of size s such that*

$$Pr_{x,r}[C(f(x), r) = \langle x, r \rangle] = \frac{1}{2} + \varepsilon.$$

Then there is a circuit $\mathcal{C}'$ of size $O((s+t)\cdot poly(n,1/\varepsilon))$ such that

$$Pr_x[\mathcal{C}'(f(x))=x]=\frac{\varepsilon}{4}.$$

Theorem 3 *If $\mathcal{VC}=(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ is a secure weak multi-function public VC scheme (see Definitions 10), then $\mathcal{ABE}_{VC}$, the ABE scheme obtained with Construction 3, is IND-CPA secure (Definition 6).*

Proof Sketch. Assume for the sake of contradiction that there exists an adversary $\mathcal{A}_{ABE}$ that wins with non-negligible probability μ the security game from Definition 6. We use this to break the soundness of the VC protocol in two steps, conceptually.

First, using Lemma 1, the existence of $\mathcal{A}_{ABE}$ means that there is an adversary that, given a ciphertext of the form (σ_x, r, D), predicts an inverse of VK_x^1 under the function H. Note that this transformation creates an adversary $\mathcal{A}'_{ABE}$ that asks polynomially many more ABE secret key queries than $\mathcal{A}_{ABE}$.

Since this adversary essentially predicts the message of the server, this can then be used to construct an adversary $\mathcal{A}_{VC}$ that breaks the soundness of the VC protocol (from Definition 10) with non-negligible probability. For completeness, we describe both these transformations as one algorithm $\mathcal{A}_{VC}$ that uses $\mathcal{A}_{ABE}$:

1. $\mathcal{A}_{VC}$ receives PK_{param}, the output from $\mathcal{VC}.\mathsf{Setup}(\lambda)$ from his challenger. He forwards PK_{param} to $\mathcal{A}_{ABE}$.
2. On calls to $\mathcal{O}_{ABE.\mathsf{KeyGen}}(F)$, $\mathcal{A}_{VC}$ queries his own $\mathcal{O}_{VC.\mathsf{KeyGen}}(F)$ oracle and returns the resulting PK_F.
3. Given the challenge messages (M_0, M_1) and attributes γ, $\mathcal{A}_{VC}$ requests as his challenge $(\sigma_\gamma, VK_\gamma = (VK_\gamma^0, VK_\gamma^1)) \leftarrow \mathsf{ProbGen}_{PK_{param}}(\gamma)$.
4. $\mathcal{A}_{VC}$ runs the adversary $\mathcal{A}_{gl}$ to obtain the next value r submitted to the oracle $\mathcal{O}_x(\cdot)$. $\mathcal{A}_{VC}$ chooses a random bit $d \overset{R}{\leftarrow} \{0,1\}$ and returns the challenge ciphertext $C=(r,\sigma_\gamma,d)$ to $\mathcal{A}_{ABE}$.
5. Eventually, $\mathcal{A}_{ABE}$ will return his guess bit $\hat{b}$. $\mathcal{A}_{VC}$ computes $d \oplus M_{\hat{b}}$ and returns this value as an answer to $\mathcal{A}_{gl}$.
6. When $\mathcal{A}_{gl}$ makes a new oracle query, $\mathcal{A}_{VC}$ rewinds the execution of $\mathcal{A}_{ABE}$ to Step 4.
7. When $\mathcal{A}_{VC}$ receives an answer $\tilde{\sigma}_0$ from $\mathcal{A}_{gl}$, he returns it as his output for the computation on input $(\sigma_\gamma, VK_\gamma = (VK_\gamma^0, VK_\gamma^1))$.

The probability that $\mathcal{A}_{VC}$ succeeds in cheating is the same as the probability that $\mathcal{A}_{gl}$ succeeds in inverting $VK_\gamma^0 = H(\sigma_0)$. By assumption $\mathcal{A}_{ABE}$ wins the security game from Definition 6 with non-negligible probability μ. Therefore $\mathcal{A}_{VC}$ returns the correct value for $\sigma_0 \cdot r$ to $\mathcal{A}_{gl}$ with probability μ. Then by Lemma 1 it follows that $\mathcal{A}_{gl}$ will compute the correct output σ_0 with probability $\frac{\varepsilon}{4}$, which is non-negligible. Hence $\mathcal{A}_{VC}$ wins the security game from Definition 10 with non-negligible probability. Also $\mathcal{A}_{VC}$ runs in polynomial time since $\mathcal{A}_{gl}$ is a polynomial-time algorithm. ■