Verifiable Delegation of Computation over Large Datasets

Siavosh Benabbas* Rosario Gennaro† Yevgeniy Vahlis‡

July 11, 2011

Abstract

We study the problem of computing on large datasets that are stored on an untrusted server. We follow the approach of *amortized verifiable computation* introduced by Gennaro, Gentry, and Parno in CRYPTO 2010. We present the first practical verifiable computation scheme for high degree polynomial functions. Such functions can be used, for example, to make predictions based on polynomials fitted to a large number of sample points in an experiment. In addition to the many non-cryptographic applications of delegating high degree polynomials, we use our verifiable computation scheme to obtain new solutions for verifiable keyword search, and proofs of retrievability. Our constructions are based on the DDH assumption and its variants, and achieve adaptive security, which was left as an open problem by Gennaro *et al* (albeit for general functionalities).

Our second result is a primitive which we call a *verifiable database* (VDB). Here, a weak client outsources a large table to an untrusted server, and makes retrieval and update queries. For each query, the server provides a response and a proof that the response was computed correctly. The goal is to minimize the resources required by the client. This is made particularly challenging if the number of update queries is unbounded. We present a VDB scheme based on the hardness of the subgroup membership problem in composite order bilinear groups. This is the first such construction that relies on a "constant-size" assumption, and does not require expensive generation of primes per operation.

*University of Toronto, siavosh@cs.toronto.edu

†IBM Research, rosario@us.ibm.com

‡Columbia University, evahlis@cs.columbia.edu

Contents

1 Introduction

This paper presents very efficient protocols that allow a computationally weak client to securely outsource some computations over very large datasets to a powerful server. Security in this context means that the client will receive an assurance that the computation performed by the server is correct, with the optional property that the client will be able to hide some of his data from the server.

The problem of securely outsourcing computation has received widespread attention due to the rise of *cloud computing:* a paradigm where businesses lease computing resources from a service (the *cloud provider*) rather than maintain their own computing infrastructure [2, 60]. A crucial component of secure cloud computing is a mechanism that enforces the integrity and correctness of the computations done by the provider.

Outsourced computations are also increasingly relevant due to the proliferation of mobile devices, such as smart phones and netbooks, computationally weak devices which might off-load heavy computations, e.g., a cryptographic operation or a photo manipulation, to a network server. Here too, a proof of the correctness of the result might be desirable if not necessary.

A crucial requirement in all of these cases is that the computation invested by the (weak) client in order to verify the result of the server's work must be substantially smaller than the amount of computation required to perform the work to begin with. Indeed if that was not the case, the client could perform the computation on its own without interacting with the server! It is also desirable to keep the server's overhead as small as possible: in other words the computation of the server to provide both the result and a correctness proof to the client should be as close as possible to the amount of work needed to simply compute the original function (otherwise, the server, which might provide this service to many clients, may become overwhelmed by the computational load).

This paper initiates a line of research about efficient protocols for verifiable computation of specific functions, in our case the evaluation of polynomials derived from very large datasets. Most of the prior work (reviewed below) has focused on generic solutions for arbitrary functions. So while in "general" the problem we are considering has been solved, by focusing on specific computations we are able to obtain much more efficient protocols. This is similar to the way research over secure multiparty computation has evolved: following generic protocols for the evaluation of arbitrary functions [63, 29, 9, 17], there has been a twenty-plus year effort to come up with efficient distributed protocols for specific computations encountered in practical applications (e.g. the entire work on threshold cryptography [21], or protocols on set intersection and pattern matching such as [34]).

OUR RESULTS. This paper focuses on the evaluation of polynomials derived from very large datasets. While the computations themselves are simple, it's the magnitude of data that prevents the client (who cannot even store the entire data) to perform them by itself. In our protocols the client will initially store the data at the server (with the option of encrypting it for confidentiality, if desired), with some authenticating information. The client will only keep a short secret key. Later, every time the client requests the value of a computation over the data, the server will compute the result and return it together with an *authentication code*, which the client will be able to quickly verify with the secret key. This description shows that our problem naturally fits into the *amortized* model for outsourced computation introduced in [27]: the client performs a one-time computationally expensive phase (in our case storing the data with its authentication information) and then quickly verifies the results provided by the server.

Our protocols are very efficient. The computation of the authentication data is comparable to encrypting the file using the ElGamal encryption scheme (i.e. roughly 2 exponentiations per data block). Verification takes at most a logarithmic (in the number of blocks) number of exponentiations under the DDH Assumption. Additionally, we present a faster protocol (which requires only a *single* exponentiation to verify the

result) which is secure under a decisional variant of the Strong Diffie Hellman Assumption in single groups[1].

An immediate application of our results is the ability to verifiably outsource computations to make predictions based on polynomials fitted to a large number of sample points in an experiment.

In the second part of our paper, we present an extension to our protocols, which allows the client to efficiently *update* the data (and its associated authentication information) stored at the server. We also present applications of our protocols to the problems of verifiable *keyword search* (the client stores a large database with the server and it queries if a specific keyword appears in it) and secure *proofs of retrievability* (the client checks that the file stored with the server is indeed retrievable) [53, 36].

VERIFIABLE DELEGATION OF POLYNOMIALS. The basis of all our protocols is verifiable delegation of polynomials. Assume the client has a polynomial $P(\cdot)$ of large degree d, and it wants to compute the value $P(x)$ for an arbitrary inputs x. In our basic solution the client stores the polynomial in the clear with the server as a vector $\boldsymbol{c}$ of coefficients in $\mathbb{Z}_p$. The client also stores with the server a vector $\boldsymbol{t}$ of group elements of the form $g^{ac_i+r_i}$ where $a \in_{\mathrm{R}} \mathbb{Z}_p$ and r_i is the i^{th}-coefficient of a polynomial $R(\cdot)$ of the same degree as $P(\cdot)$. When queried on input x the server returns $y = P(x)$ and $t = g^{aP(x)+R(x)}$ and the client accepts y iff $t = g^{ay+R(x)}$.

If $R(\cdot)$ was a random polynomial, then we can prove that this is a secure delegation scheme in the sense of [27]. However checking that $t = g^{ay+R(x)}$ would require the client to perform computation polynomial in the degree of $P(\cdot)$ – the exact work that we set out to avoid. The crucial point, therefore, is how to perform this verification fast, in time which is independent, or at the very least sublinear in the degree of $P(\cdot)$. We do that by defining $r_i = F_K(i)$ where F is a pseudo-random function (PRF in the following) with a special property which we call *closed form efficiency*. The property is that given the polynomial $R(\cdot)$ defined by the r_i coefficients, the value $R(x)$ (for any input x) can be computed very efficiently (sub-linearly in d) by a party who knows the secret key K for the PRF. Since F is a PRF, the security of the scheme is not compromised (as F is indistinguishable from a random function), and the closed form efficiency of F will allow the client to verify the result in time sub-linear in the degree of the polynomial.

We generalize our result for PRFs with other types of closed form efficiency, which yield efficient and secure delegation protocols not only for single-variable polynomials of degree d, but also for multivariate polynomials with total degree d or of degree d in each variable. We have several different variations of PRFs: the least efficient one is secure under the Decisional Diffie-Hellman assumption, while more efficient ones require a decisional variant of the Strong DH assumption.

Adaptivity: One of the main questions to remain open after the work of GGP [27] is whether we can achieve verifiable delegation even if the malicious server knows whether the verifier accepted or rejected the correctness proof of the value computed by the server. Indeed, the GPV scheme becomes insecure if the server learns this single bit of information after each proof is sent to the verifier. Our constructions are the first to achieve adaptive security in the amortized setting.

Privacy: Our solution allows the client to preserve the secrecy of the polynomial stored with the server, by encrypting it with an additively homomorphic encryption scheme. In this case the server returns an encrypted form of y which the client will decrypt.

Keyword Search: The applications to keyword search without updates is almost immediate. Consider a text file $F = \{w_1, \ldots, w_\ell\}$ where w_i are the words contained in it. Encode F as the polynomial $P(\cdot)$ of degree ℓ such that $P(w_i) = 0$. To make this basic solution efficiently updatable we use a variation of the polynomial delegation scheme which uses bilinear maps. We also present a generic, but less efficient way to make *any* static keyword search protocol updatable which might be of independent interest.

Proof of Retrievability: Again the application of our technique is quite simple. The client encodes the file as a polynomial $F(x)$ of degree d (each block representing a coefficient), and delegates the computation of

[1] See e.g., [19] for a survey of the strong DH family of assumptions

$F(x)$ to the server. The proof of retrievability consists of the client and the server engaging in our verifiable delegation protocol over a random point r: the client sends r and the server returns the value $F(r)$ together with a proof of its correctness. The client accepts if it accepts the proof that $F(r)$ is the correct value.

VERIFIABLE DATABASES WITH EFFICIENT UPDATES. In the second part of our paper we study the problem of verifiable databases, where a resource constrained client wishes to store an array DB on a server, and to be able to retrieve the value at any cell $DB[i]$, and to update the database by assigning $DB[i] \leftarrow v$ for a new value v. The goal is to achieve this functionality with an additional guarantee that if a server attempts to tamper with the data, the tampering will be detected when the client queries the database.

Simple solutions (based on Message Authentication Codes or Signature Schemes) exist for the restricted case where the database is static – i.e. the client only needs to retrieve data, but does not modify the database. One example is to have the client sign each pair (index,value) that is sent to the server. Clearly, if no updates are performed, the server has no choice but to return the correct value for a given cell. However, the problem becomes significantly harder when efficient updates are needed. One solution is for the client to just keep track of all the changes locally, and apply them as needed, but this contradicts our goal of keeping client state and workload as small as possible. On a high level, the main technical difficulty stems from the fact that the client must revoke any authenticating data that the server has for the previous value of the updated cell. This issue has been addressed in the line of works on cryptographic accumulators [16, 49, 56], and, using different techniques, in the authenticated datastructures literature [50, 43, 55, 62].

We present a verifiable database delegation scheme based on the hardness of the subgroup membership problem in composite order bilinear groups (this assumption was originally introduced in [13]). Our solution allows the client to query any location of the database, and verify the response in time that is independent of the size of the database. The main advantage of our construction is that it allows the client to insert and delete values, as well as update the value at any cell by sending a single group element to the server after retrieving the current value stored in the cell. Prior solutions either rely on non-constant size assumptions (such as variants of the Strong Diffie-Hellman assumption [23, 15]), require expensive generation of primes for each operation (in the worst case), or require expensive "re-shuffling" procedures to be performed once in a while on the data. On the other hand, our construction works in the private key setting, whereas some prior solutions allow public verification (e.g., [16, 49]).

ROADMAP. The rest of the paper is organized as follows. In Section 2 we define the security assumptions used in the paper. Readers interested in the precise definition of Verifiable Computation and its security can find them in Section 3. In Section 4 we introduce our notation of Algebraic Pseudorandom Functions which are the main building block of our constructions. In Section 5 we show how to securely delegate polynomial evaluations to an untrusted server using Algebraic PRFs. In Section 6 we use delegation of polynomials to implement verifiable databases. Finally our results on Proof of Retrievability are given in Section 7.

1.1 Related Work

As mentioned above our work follows the paradigm introduced in [27] which is also adopted in [20, 3]. The protocols described in those papers allow a client to outsource the computation of an arbitrary function (encoded as a Boolean circuit) and use fully homomorphic encryption (i.e. [28]) resulting in protocols of limited practical relevance. Our protocols on the other hand work for only a very limited class of computations (mostly polynomial evaluations) but are very efficient and easily implementable in practice.

The previous schemes based on fully homomorphic encryption also suffer from the following drawback: if a malicious server tries to cheat and learns if the client has accepted or rejected its answer, then the client must repeat the expensive pre-processing stage. The only alternative way to deal with this problem proposed in these papers is to protect this bit of information from the server (which is a very strong assumption to make). Somewhat surprisingly our scheme remains secure even if a cheating server learns the

acceptance/rejection bit of the client, without any need to repeat the pre-processing stage. This is not only conceptually interesting, but also a very practical advantage.

There is a large body of literature, prior to [27], that investigates the problem of verifiably outsourcing the computation of an arbitrary functions (we refer to [27] for an exhaustive list of citations). This problem has attracted the attention of the Theory community, starting from the work on Interactive Proofs [5, 31], efficient arguments based on probabilistically checkable proofs (PCP) [37, 38], CS Proofs [45] and the *muggles proofs* in [30]. However in PCP-based schemes, the client must store the large data in order to verify the result and therefore these solutions might not be applicable to our setting.

This problem has also been studied by the Applied Security community, with solutions which are practical but whose security holds under some very strong assumptions on the behavior of the adversary. For example, solutions based on audit (e.g. [48, 6]) which typically assume many clients, and require a fraction of them to recompute some of the results provided by the server, but are secure only under the assumption that bad clients do not collude. Another approach is to use secure co-processors (e.g. [59, 64]) which "sign" the computation as correct, under the assumption that the adversary cannot tamper with the processor. Finally, other trust models have been considered. The area of authenticated data structures [61, 43, 57] aims to provide delegation solutions for data storage, modification, and retrieval. These works additionally study a three-party model, where the owner maintains a large state, and acts as a trusted third party, but delegates his data to an untrusted server that can be queried by weak clients.

For the specific case of outsourcing expensive cryptographic operations, Chaum and Pedersen in [18], describe protocols to allow a client to verify the behavior of a piece of hardware placed on the client's device by a service provider such as a bank. Hohenberger and Lysyanskaya formalize this model [35], and present protocols for the computation of modular exponentiations (arguably the most expensive step in public-key cryptography operations). Their protocol requires the client to interact with *two* non-colluding servers. Other work targets specific function classes, such as one-way function inversion [32].

The application of secure keyword search over a stored file can be handled using *zero-knowledge sets*, [46] which however does not allow for an easy way to update the file. Our protocol for keyword search combines ideas from our polynomial delegation scheme with some machinery inspired by zero-knowledge sets, to obtain a protocol that allows for efficient updates and other additional desirable properties (see Section 6.1).

The problem of proof of retrievability was first posed in [53, 36], and subsequent protocols include [4, 58, 22]. A proof of retrievability protocol usually goes like this: after storing a (potentially large) file with the server, the client issues a *query* to receive an assurance that the file is still correctly stored. The server computes an answer based on the query and the file, and finally the client performs some verification procedure on the answer. All of these protocols incur a substantial storage overhead for the server (since the file is stored using an erasure code) and, except for [22], require communication which is quadratic in the security parameter. The protocol in [22] has linear communication complexity but it requires *both* the server and the client to work in time proportional to the size of the file. Our solution achieves linear communication complexity in the security parameter and is very efficient for the client (as its work is sublinear in the size of the file).

Our verifiable database construction is closely related to Memory Checkers (see e.g. [10, 26, 1, 24, 53]). However, our setting differs from the memory checking setting in that we allow the server to be an arbitrary algorithm, whereas a memory checker interacts with a RAM (an oracle that accepts store/retrieve queries). In this context, our construction would yield a memory checker with poor performance since it would require the checker to issue a number of queries that is linear in the size of the memory. In contrast, we focus on optimizing the communication and the work of the client when the server can perform arbitrary computation on its data. Our construction requires the server to perform a linear amount of work to answer one type of queries (update/retrieve), while the other type of queries requires only a constant amount of work. Finally,

we note that the work on accumulators [16, 49, 56] and authenticated data structures [50, 43, 55, 62] can be used to construct verifiable databases with similar efficiency under "non-constant size" assumptions (such as Strong Diffie-Hellman), or under the strong RSA assumption, but requiring generation of random primes to perform certain update operations. We direct the reader to [56] for a good survey of accumulator based data structures.

2 Assumptions

In this work we rely on the following assumptions about computational groups.

DECISIONAL DIFFIE HELLMAN. The standard Decisional Diffie-Hellman Assumption (DDH) is defined as follows. For every PPT distinguisher A there exists a negligible function $neg(\cdot)$ such that for all n,

$$|\Pr[A(1^n, g, g^x, g^y, g^{xy}) = 1] - \Pr[A(1^n, g, g^x, g^y, g^z) = 1]| \leq neg(n)$$

where g is a generator of a group $\mathbb{G}$ of order p where p is a prime of length approximately n, and $x, y, z \in_{\mathrm{R}} \mathbb{Z}_p$.

STRONG DIFFIE HELLMAN. The strong Diffie-Hellman family of assumptions allows an adversary to obtain group elements $g, g^x, g^{x^2}, \ldots, g^{x^d}$, and requires the adversary to compute or distinguish a related group element from a random one. Computational variants of the problem appeared as early as the work of Mitsunari *et al* [47]. More recently, bilinear versions of the assumptions, starting with the works of Boneh and Boyen [11, 12], were used in several applications (e.g. [23, 15]). Boneh and Boyen gave a proof of the bilinear assumptions in the generic group model. In one of our constructions, we achieve high efficiency by relying on a decisional version of the strong DH assumption in single groups.

The d-SDDH assumption is stated as follows. For every PPT distinguisher A there exists a negligible function $neg(\cdot)$ such that for all n,

$$|\Pr[A(1^n, g, g^x, g^{x^2}, \ldots, g^{x^d}) = 1] - \Pr[A(1^n, g, g^{x_1}, g^{x_2}, \ldots, g^{x_d}) = 1]| \leq neg(n)$$

where g is a generator of a group $\mathbb{G}$ of order p where p is a prime of length approximately n, and $x, x_1, \ldots, x_d \in_{\mathrm{R}} \mathbb{Z}_p$.

SUBGROUP MEMBERSHIP ASSUMPTION IN COMPOSITE ORDER BILINEAR GROUPS. The subgroup membership assumption in composite order bilinear groups first appeared in [13], and has seen many recent applications in the areas of Identity Based Encryption (IBE), Hierarchical IBE, and others [25, 13, 8, 39, 42]. The assumption we rely on (for our verifiable database delegation scheme) is the following.

For every PPT distinguisher A there exists a negligible function $neg(\cdot)$ such that for all n,

$$|\Pr[A(1^n, g_1 g_2, u_2, (g_1 g_2)^x) = 1] - \Pr[A(1^n, g_1 g_2, u_2, u_2^x) = 1]| \leq neg(n)$$

where $\mathbb{G}$ is a group of order $N = p_1 p_2$ where p_1 and p_2 are primes of length approximately n, $\mathbb{G}_1$ and $\mathbb{G}_2$ are subgroups of $\mathbb{G}$ of orders p_1 and p_2 respectively, $g_1 \in_{\mathrm{R}} \mathbb{G}_1$, $g_2, u_2 \in_{\mathrm{R}} G_2$, and $x \in_{\mathrm{R}} \mathbb{Z}_N$.

In addition, we require the existence of an efficiently computable pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ where $\mathbb{G}_T$ is a group of order N. We shall make use of the following property of pairings over composite order groups: for $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, $e(g_1, g_2) = 1_{\mathbb{G}_T}$. This property holds for every bilinear pairing over composite order groups (as shown e.g. in [40]).

BILINEAR SUB-GROUP PROJECTION ASSUMPTION. In the analysis of our verifiable database scheme we first show the security of the scheme based on the following new assumption. We then apply Lemma

2.1 (given below) to obtain a reduction to the subgroup membership problem. The Bilinear Sub-Group Projection Assumption (BSGP) is stated as follows: for every PPT adversary A, there exists a negligible function $neg(\cdot)$ such that for all n,

$$\Pr[A(1^n, (g_1 g_2), (h_1 h_2), u_2) = e(g_1, h_1)] \leq neg(n)$$

where $\mathbb{G}$ is a group of order $N = p_1 p_2$ where p_1 and p_2 are primes of length approximately n, $\mathbb{G}_1$ and $\mathbb{G}_2$ are subgroups of $\mathbb{G}$ of orders p_1 and p_2 respectively, $g_1, h_1 \in_R \mathbb{G}_1$, and $g_2, h_2, u_2 \in_R G_2$. The following lemma shows that the BSGP assumption is implied by the standard sub-group membership assumption in composite order bilinear groups.

Lemma 2.1. *The subgroup membership assumption in composite order bilinear groups reduces to the BSGP assumption.*

Proof. Let A be an adversary breaking the BSGP assumption with advantage ε. We construct a distinguisher B that distinguishes between a random member of $\mathbb{G}$ and a random member of $\mathbb{G}_2$ with advantage $\varepsilon^3/64$. B is given a challenge t, and generators $g_{12} = g_1 g_2$ of $\mathbb{G}$, and u_2 of $\mathbb{G}_2$.

Intuitively, B is going to run use A to compute the pairing e both as $e(g_1, h_1)$ and $e(g_1 t_1, h_1)$. To do this it will run A on (g_{12}, h_{12}) and will check if its output will change when the first operand is multiplied by t. If $t \in \mathbb{G}_2$ then output should not change. The issue with this intuition is that A can be wrong on many inputs and in fact it might have a few outputs which are very likely whenever it is wrong, e.g. A might output $1_{\mathbb{G}_T}$ whenever it does not find a suitable output. To account for this our adversary will first run A a few times and collects statistics about its very likely outputs and later ignores these outputs of A.

The precise implementation is as follows. We will use a parameter $\delta = \frac{\varepsilon^3}{64 \ln^2 n}$; This choice will become clear later. B starts by generating $h_{12} = h_1 h_2 = g_{12}^{\alpha}$ by choosing $\alpha \in_R \mathbb{Z}_N$. It then repeats the following procedure $1/\delta$ times: generate $\beta \in_R \mathbb{Z}_N$, and run $A(1^n, g_{12}^{\beta}, h_{12}, u_2)$ to obtain a value v. Let V be the set of all values obtained during this procedure.

After collecting the above samples, B runs $A(1^n, g_{12}, h_{12}, u_2)$ and $A(1^n, g_{12} \cdot t, h_{12}, u_2)$ to obtain values w and w' respectively. If $\{w, w'\} \cap V \neq \emptyset$ or if $w \neq w'$, B outputs 0. Otherwise, we have that $w = w'$ and $w \notin V$. In this case, B outputs 1.

Our next step is to analyze B's probability of success. Let us first consider the case that $t \in_R \mathbb{G}_2$. We denote by E the event that $A(1^n, g_{12}, h_{12}, u_2)$ "wins" and outputs the correct value $e(g_1, h_1)$. Note that it may not be easy to determine whether E occurred. For notational clarity, in the following we assume that A is deterministic (the analysis extends to the randomized case). We define $E[g_{12}, h_{12}, u_2]$ to be the event E when g_{12}, h_{12}, u_2 are the inputs to A.

Suppose that both $E[g_{12} \cdot t, h_{12}, u_2]$ and $E[g_{12}, h_{12}, u_2]$ occur. Then, we have $w = e(g_1, h_1)$ and $w' = e(g_1 t_1, h_1)$, where t_1 is the $\mathbb{G}_1$ component of t, which is $1_{\mathbb{G}}$ when $t \in \mathbb{G}_2$. Therefore, in this case, the equality $w = w'$ holds. This implies,

$$\begin{aligned}
&\Pr[B(1^n, t, g_{12}, h_{12}, u_2) = 1 | t \in_R \mathbb{G}_2] = \Pr\left[(w = w') \wedge (w \notin V) | t \in_R \mathbb{G}_2\right] \\
&\geq \Pr\left[E[g_{12} \cdot t, h_{12}, u_2] \wedge E[g_{12}, h_{12}, u_2] \wedge e(g_1, h_1) \notin V | t \in_R \mathbb{G}_2\right] \\
&\geq \Pr\left[E[g_{12} \cdot t, h_{12}, u_2] \wedge E[g_{12}, h_{12}, u_2] | t \in_R \mathbb{G}_2\right] - \Pr\left[e(g_1, h_1) \in V | t \in_R \mathbb{G}_2\right] \qquad (1)
\end{aligned}$$

Let us now analyze the probability that both $E[g_{12} \cdot t, h_{12}, u_2]$ and $E[g_{12}, h_{12}, u_2]$ occur. To do so, we show that the value $\Pr_{g_2}[E[g_1 g_2, h_{12}, u_2] | g_1, h_{12}, u_2]$ is unlikely to be too low. First observe that the following holds

$$\Pr_{g_1, h_{12}, u_2}\left[\Pr_{g_2}\left[E[g_1 g_2, h_{12}, u_2] | g_1, h_{12}, u_2\right] \geq \varepsilon/4\right] > 3\varepsilon/4,$$

as otherwise $\Pr[E] \leq \frac{\varepsilon}{4}(1-3\varepsilon) + \frac{3\varepsilon}{4} < \varepsilon$ which is a contradiction. Now observe that once we condition on g_1, h_{12}, u_2, the two events $E[g_{12}, h_{12}, u_2]$ and $E[g_{12} \cdot t, h_{12}, u_2]$ are independent and have the same probability. Plugging these two observations into (1) implies,

$$\begin{aligned}
&\Pr[B(1^n, t, g_{12}, h_{12}, u_2) = 1 | t \in_{\mathrm{R}} \mathbb{G}_2] \\
&\geq \mathbb{E}_{g_1, h_{12}, u_2}\left[\Pr_{g_2, t}\left[E[g_{12} \cdot t, h_{12}, u_2] | g_1, h_{12}, u_2\right] \Pr_{g_2, t}\left[E[g_{12}, h_{12}, u_2] | g_1, h_{12}, u_2\right] \middle| t \in_{\mathrm{R}} \mathbb{G}_2\right] \\
&\quad - \Pr\left[e(g_1, h_1) \in V | t \in_{\mathrm{R}} \mathbb{G}_2\right] \\
&\geq \mathbb{E}_{g_1, h_{12}, u_2}\left[\Pr_{g_2}\left[E[g_{12} \cdot t, h_{12}, u_2] | g_1, h_{12}, u_2\right]^2 \middle| t \in_{\mathrm{R}} \mathbb{G}_2\right] - \Pr\left[e(g_1, h_1) \in V | t \in_{\mathrm{R}} \mathbb{G}_2\right] \\
&\geq 3\varepsilon/4 \times (\varepsilon/4)^2 - \Pr\left[e(g_1, h_1) \in V | t \in_{\mathrm{R}} \mathbb{G}_2\right] \geq 3\varepsilon^3/64 - \Pr\left[e(g_1, h_1) \in V | t \in_{\mathrm{R}} \mathbb{G}_2\right] \qquad (2)
\end{aligned}$$

Let us now analyze the probability that $e(g_1, h_1) \in V$. We rely on the fact that that after fixing h_1, $e(g_1, h_1)$ is uniformly distributed over a subset of $\mathbb{G}_T$ of size p_1 and that $|V| = 1/\delta$. Therefore, (2) reduces to,

$$\Pr[B(1^n, t, g_{12}, h_{12}, u_2) = 1 | t \in_{\mathrm{R}} \mathbb{G}_2] \geq 3\varepsilon^3/64 - |V|/p_1 \geq 3\varepsilon^3/64 - \frac{1}{\delta p_1}. \qquad (3)$$

We now turn to analyze the case $t \in \mathbb{G}$. In this case, we observe that (once we choose h_{12} and u_2, w and w' are independent and distributed identically to the $\lceil 1/\delta \rceil$ samples $V = \left\{A(1^n, g_{12}^{\beta_i}, h_{12}, u_2)\right\}_{i=1}^{\lceil 1/\delta \rceil}$ generated by B prior to computing w, w'. Now consider the value w. If $\Pr_{g_{12}}[A(1^n, g_{12}^{\beta_i}, h_{12}, u_2) = w] \geq \delta \ln^2 n$ then $\Pr[w \notin V] \leq (1-\delta n)^{1/\delta} \leq e^{-\ln^2 n}$. Hence, we get

$$\Pr[B(1^n, t, g_{12}, h_{12}, u_2) = 1 | t \in_{\mathrm{R}} \mathbb{G}] = \Pr[\{w, w'\} \cap V = \emptyset \wedge w = w'] \leq \frac{1}{n^{\ln n}} + \delta \ln^2 n. \qquad (4)$$

Combining (3) and (4),

$$\begin{aligned}
&|\Pr[B(1^n, t, g_{12}, h_{12}, u_2) = 1 | t \in_{\mathrm{R}} \mathbb{G}_2] - \Pr[B(1^n, t, g_{12}, h_{12}, u_2) = 1 | t \in_{\mathrm{R}} \mathbb{G}]| \\
&\qquad \geq 3\varepsilon^3/64 - \frac{1}{\delta p_1} - \frac{1}{n^{\ln n}} - \delta \ln^2 n > \varepsilon^3/64,
\end{aligned}$$

where the inequality follows because the second and third terms are (super-polynomially) small and δ is chosen so that the last term is at most $\varepsilon^3/64$. □

3 Verifiable Computation

A verifiable computation scheme is a two-party protocol between a *client* and a *server*. The client chooses a function and an input which he provides to the server. The latter is expected to evaluate the function on the input and respond with the output together with a *proof* that the result is correct. The client then verifies that the output provided by the worker is indeed the output of the function computed on the input provided.

The goal of a verifiable computation scheme is to make such verification very efficient, and particularly much faster than the computation of the function itself. We adopt the *amortized* model of Gennaro *et al.* [27]: for each function F, the client is allowed to invest a one-time expensive computational effort (comparable to the effort to compute F itself) to produce a public/secret key pair, which he will use to efficiently (e.g. in linear-time) verify the computation of F by the server on many inputs.

We prove our results in a stronger version of the [27] definition of verifiable computation scheme. As we discussed in the Introduction, the main difference is that in our protocols the server is allowed to learn if the

client accepts or rejects the output of a particular computation (in [27] and following works in the amortized model, leaking this bit of information to the server would help him cheat in the following executions).

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

1. $\mathbf{KeyGen}(f, n) \rightarrow (PK, SK)$: Based on the security parameter n, the randomized *key generation* algorithm generates a public/secret key pair for the function f. The public key is provided to the server, while the client keeps the matching secret key private.

2. $\mathbf{ProbGen}_{SK}(x) \rightarrow (\sigma_x, \tau_x)$: The *problem generation* algorithm uses the secret key SK to encode the function input x as a public value σ_x which is given to the server, and a secret value τ_x which is kept private by the client.

3. $\mathbf{Compute}_{PK}(\sigma_x) \rightarrow \sigma_y$: Using the client's public key for f and the encoded input, the server *computes* an encoded version of the function's output $y = f(x)$.

4. $\mathbf{Verify}_{SK}(\tau_x, \sigma_y) \rightarrow y \cup \perp$: Using the secret key SK and the secret "decoding" value τ_x, the *verification* algorithm converts the worker's encoded output into the output of the function, e.g., $y = f(x)$ or outputs $\perp$ indicating that σ_y does not represent the valid output of f on x.

SECURITY. A verifiable computation scheme should be both correct and secure. A scheme is correct if the problem generation algorithm produces values that allows an honest server to compute values that will verify successfully and correspond to the evaluation of f on those inputs. More formally, let $\mathcal{F}$ be a family of functions:

Definition 3.1 (Correctness)**.** A verifiable computation scheme $\mathcal{VC}$ is *$\mathcal{F}$-correct* if for any choice of function $f \in \mathcal{F}$, the key generation algorithm produces keys $(PK, SK) \leftarrow \mathbf{KeyGen}(f, \lambda)$ such that, $\forall x \in \texttt{Domain}(f)$, if $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{SK}(x)$ and $\sigma_y \leftarrow \mathbf{Compute}_{PK}(\sigma_x)$ then $y = f(x) \leftarrow \mathbf{Verify}_{SK}(\tau_x, \sigma_y)$.

Intuitively, a verifiable computation scheme is secure if a malicious worker cannot persuade the verification algorithm to accept an incorrect output. In other words, for a given function f and input x, a malicious worker should not be able to convince the verification algorithm to output $\hat{y}$ such that $f(x) \neq \hat{y}$. Below, we formalize this intuition with an experiment, where $poly(\cdot)$ is a polynomial.

Experiment $\mathbf{Exp}_A^{Verif}[\mathcal{VC}, f, n]$
$(PK, SK) \leftarrow \mathbf{KeyGen}(f, n)$;
For $i = 1, \ldots, \ell = poly(n)$;
$x_i \leftarrow A(PK, x_1, \sigma_{x,1}, \beta_1, \ldots, x_{i-1}, \sigma_{x,i-1}, \beta_{i-1})$;
$(\sigma_{x,i}, \tau_{x,i}) \leftarrow \mathbf{ProbGen}_{SK}(x_i)$;
$\sigma_{y,i} \leftarrow A(PK, x_1, \sigma_{x,1}, \beta_1, \ldots, x_{i-1}, \sigma_{x,i-1}, \beta_{i-1}, \sigma_{x,i})$;
$\beta_i \leftarrow \mathbf{Verify}_{SK}(\tau_{x,i}, \sigma_{y,i})$;
$x \leftarrow A(PK, x_1, \sigma_{x,1}, \beta_1, \ldots, x_\ell, \sigma_{x,\ell}, \beta_\ell)$;
$(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{SK}(x)$;
$\hat{\sigma}_y \leftarrow A(PK, x_1, \sigma_{x,1}, \beta_1, \ldots, x_\ell, \sigma_{x,\ell}, \beta_\ell, \sigma_x)$;
$\hat{y} \leftarrow \mathbf{Verify}_{SK}(\tau_x, \hat{\sigma}_y)$
If $\hat{y} \neq \perp$ and $\hat{y} \neq f(x)$, output '1', else '0';

Essentially, the adversary is given oracle access to generate the encoding of multiple problem instances, and also oracle access to the result of the verification algorithm on arbitrary strings on those instances. The adversary succeeds if it convinces the verification algorithm to accept on the wrong output value for a given input value. Our goal is to make the adversary succeed only with negligible probability.

Definition 3.2 (Security)**.** For a verifiable computation scheme $\mathcal{VC}$, we define the advantage of an adversary A in the experiment above as:

$$Adv_A^{Verif}(\mathcal{VC}, f, n) = Prob[\mathbf{Exp}_A^{Verif}[\mathcal{VC}, f, n] = 1] \tag{5}$$

A verifiable computation scheme $\mathcal{VC}$ is *$\mathcal{F}$-secure* if for any function $f \in \mathcal{F}$, and for any adversary A running in probabilistic polynomial time,

$$Adv_A^{Verif}(\mathcal{VC}, f, \lambda) \leq \texttt{negli}(n) \tag{6}$$

where $\texttt{negli}()$ is a negligible function of its input.

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

Definition 3.3 (Efficiency)**.** A $\mathcal{VC}$ is efficient if for any x and any σ_y, the time required to compute $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{SK}(x)$ plus the time required for $\mathbf{Verify}(\tau_x, \sigma_y)$ is $o(T)$, where T is the time required to compute $f(x)$.

Notice that, following [27], we are not including the time to compute the key generation algorithm (i.e., the encoding of the function itself). As we said before, computing the public/private keys is a one-time operation (per function) which can then be amortized over the cost of many computations.

UPDATING THE PUBLIC KEY. In many situations it may be necessary for the client to modify the delegated function f that is stored on the server. One simple way to achieve this is to store a signed encryption of the description of f on the server. Then, to update f, the client retrieves the description, modifies it appropriately, and generates a new public key using $\mathbf{KeyGen}$. This solution has clear drawbacks, most notably the amount of communication and the amount of computation that the client needs to do.

4 Algebraic Pseudorandom Functions

Our main technical tool is a new way of viewing pseudo-random functions (PRF) with algebraic properties to achieve efficient verification of server certificates in the delegation setting. Intuitively, we rely on the fact that certain pseudo-random functions (such as the Naor-Reingold PRF [51]) have outputs that are members of an abelian group, and that certain algebraic operations on these outputs can be computed significantly more efficiently if one possesses the key of the pseudo-random function that was used to generate them. In this section we present an abstraction of the said property, and several constructions achieving different trade-offs between the types of functions that can be efficiently evaluated given the key, and the assumption that is needed to guarantee pseudo-randomness.

An algebraic pseudorandom function (PRF) consists of algorithms $\mathcal{PRF} = \langle \mathsf{KeyGen}, F, \mathsf{CFEval} \rangle$ where KeyGen takes as input a security parameter 1^n and a parameter $m \in \mathbb{N}$ that determines the domain size of the PRF, and outputs a pair $(K, param) \in \mathcal{K}_n$, where $\mathcal{K}_n$ is the key space for security parameter n. K is the secret key of the PRF, and $param$ encodes the public parameters. F takes as input a key K, public parameters $param$, an input $x \in \{0,1\}^m$, and outputs a value $y \in Y$, where Y is some set determined by $param$.

We require the following properties:

- *(Algebraic)* We say that $\mathcal{PRF}$ is algebraic if the range Y of $F_K(\cdot)$ for every $n \in \mathbb{N}$ and $(K, \textit{param}) \in \mathcal{K}_n$ forms an abelian group. We require that the group operation on Y be efficiently computable given *param*. We are going to use the multiplicative notation for the group operation.

- *(Pseudorandom)* $\mathcal{PRF}$ is pseudorandom if for every PPT adversary A, and every polynomial $m(\cdot)$, there exists a negligible function $neg : \mathbb{N} \to \mathbb{N}$, such that for all $n \in \mathbb{N}$:

$$|\Pr[A^{F_K(\cdot)}(1^n, \textit{param}) = 1] - \Pr[A^{R(\cdot)}(1^n, \textit{param}) = 1]| \leq neg(n)$$

  where $(K, \textit{param}) \leftarrow_{\mathrm{R}} \mathsf{KeyGen}(1^n, m(n))$, and $R : \{0,1\}^m \to Y$ is a random function.

- *(Closed form efficiency)* Let N be the order of the range sets of F for security parameter n. Let $z = (z_1, \ldots, z_l) \in (\{0,1\}^m)^l$, $k \in \mathbb{N}$, and efficiently computable $h : \mathbb{Z}_N^k \to \mathbb{Z}_N^l$ with $h(x) =< h_1(x), \ldots, h_l(x) >$. We say that (h, z) is *closed form efficient* for $\mathcal{PRF}$ if there exists an algorithm $\mathsf{CFEval}_{h,z}$ such that for every $x \in \mathbb{Z}_N^k$,

$$\mathsf{CFEval}_{h,z}(x, K) = \prod_{i=1}^{l} [F_K(z_i)]^{h_i(x)}$$

  and the running time of CFEval is polynomial in n, m, k but sublinear in l. When $z = (0, \ldots, l)$ we will omit it from the subscript, and write $\mathsf{CFEval}_h(x, K)$ instead.

The last condition (which distinguishes our notion from traditional PRFs) allows to compute a "weighted product" of l PRF values much more efficiently than by computing the l values separately and then combining them. Indeed, given *param*, h, x, and $F_K(z)$, one can always compute the value $\prod_{i=1}^{l} [F_K(z_i)]^{h_i(x)}$ in time linear in l (this follows from the algebraic property of the PRF). The purpose of the closed form efficiency requirement is therefore to capture the existence of a more efficient way to compute the same value given the secret key K.

Note that closed form efficiency can be defined for PRFs over arbitrary input spaces. In particular, it is a non-trivial condition to attain even when the input space is polynomial in the security parameter[2]. In the constructions needed for our delegation scheme, this will be the case.

4.1 Small Domain Algebraic PRFs From Strong DDH

Construction 1. Let $\mathcal{G}$ be a computational group scheme. The following construction $\mathcal{PRF}_1$ is an algebraic PRF with polynomial sized domains.

$\mathsf{KeyGen}(1^n, m)$: Generate a group description $(p, g, \mathbb{G}) \leftarrow_{\mathrm{R}} \mathcal{G}(1^n)$. Choose $k_0, k_1 \in_{\mathrm{R}} \mathbb{Z}_p$. Output $param = (m, p, g, \mathbb{G})$, $K = (k, k')$.

$F_K(x)$: Interpret x as an integer in $\{0, \ldots, D = 2^m\}$ where D is polynomial in n. Compute and output $g^{k_0 k_1^x}$.

Closed form efficiency for polynomials. We now show an efficient closed form for $\mathcal{PRF}_1$ for polynomials of the form

$$p(x) = F_K(0) + F_K(1)x + \cdots + F_K(d)x^d$$

where $d \leq D$. Let $h : \mathbb{Z}_p \to \mathbb{Z}_p^{d+1}$, be defined as $h(x) \stackrel{\mathsf{def}}{=} (1, x, \ldots, x^d)$. Then, we can define

$$\mathsf{CFEval}_h(x, K) \stackrel{\mathsf{def}}{=} g^{\frac{k_0(1-k_1^{d+1}x^{d+1})}{1-k_1 x}}$$

[2]When the input space is polynomial in the security parameter traditional PRFs exist unconditionally: if the input space has ℓ elements $\{x_1, \ldots, x_\ell\}$, define the key as ℓ random values $y_1, \ldots, y_\ell$ and $F_K(x_i) = y_i$. Notice however that this function does not have closed-form efficiency

Let us now write the $\prod_{i=0}^{d}[F_K(z_i)]^{h_i(x)}$ where $(z_0,\ldots,z_d)=(0,\ldots,d)$:

$$\prod_{i=0}^{d}[F_K(z_i)]^{h_i(x)}=\prod_{i=0}^{d}[g^{k_0k_1^i}]^{x^i}=g^{k_0\sum_{i=0}^{d}k_1^ix^i}$$

Applying the identity $\sum_{i=0}^{d}k_0k_1^ix^i=\frac{k_0(1-(k_1x)^{d+1})}{1-k_1x}$ we obtain the correctness of $\mathsf{CFEval}_h(x)$.

Theorem 4.1. *Suppose that the D-Strong DDH assumption holds. Then, $\mathcal{PRF}_1$ is a pseudorandom function.*

Proof. The input to the reduction is a description $(p,g,\mathbb{G})$ of a group, and a challenge $t_1,\ldots,t_d$ where t_i is either a random member of $\mathbb{G}$, or $g^{k_1^i}$, and $k_1\in\mathbb{Z}_p$ is randomly chosen once for the entire challenge. The reduction then chooses $k_0\in_{\mathrm{R}}\mathbb{Z}_p$, and computes the function $H(i)=t_i^{k_0}$ for $0\le i\le d$. Clearly, H is a random function if the t_i are random, and is equal to $F_K(\cdot)$ for $K=(k_0,k_1)$ if the t_i are determined by k_1. □

Construction 2. Let $\mathcal{G}$ be a computational group scheme. We define $\mathcal{PRF}_{2,d}$, for $d\in\mathbb{N}$, as follows:

$\mathsf{KeyGen}(1^n,m)$: Generate a group description $(p,g,\mathbb{G})\leftarrow_{\mathrm{R}}\mathcal{G}(1^n)$. Choose $k_0,k_1,\ldots,k_m\in_{\mathrm{R}}\mathbb{Z}_p$. Output $param=(m,p,g,\mathbb{G})$, $K=(k_0,k_1,\ldots,k_m)$.

$F_K(x)$: Interpret x as a vector $(x_1,\ldots,x_m)\in\{0,\ldots,d\}^m$. Compute and output $g^{k_0k_1^{x_1}\cdots k_m^{x_m}}$.

Closed form for m-variate polynomials of total degree at most d. We describe an efficient closed form for $\mathcal{PRF}_{2,d}$ for computing polynomials of the form

$$p(x_1,\ldots,x_m)=\sum_{\substack{i_1,\ldots,i_m\\ i_1+\cdots+i_m\le d}}F_K(i_1,\ldots,i_m)x_1^{i_1}x_2^{i_2}\cdots x_m^{i_m}.$$

Let $h:\mathbb{Z}_p^m\to\mathbb{Z}_p^l$, where $l=\binom{m+d}{d}$, be defined as

$$h(x_1,\ldots,x_m)\stackrel{\mathsf{def}}{=}\left(\binom{d}{i_1\ldots,i_m}\prod_{j=1}^{m}x_j^{i_j}\right)_{i_1+\cdots+i_m\le d}$$

Let $z=[z_1,\ldots,z_l]=[(i_1,\ldots,i_m)]_{i_1+\cdots+i_m\le d}\in\mathbb{Z}_d^{m\times l}$. We can now define

$$\mathsf{CFEval}_{h,z}(x_1,\ldots,x_m,K)\stackrel{\mathsf{def}}{=}g^{k_0(1+k_1x_1+\cdots+k_mx_m)^d}$$

Correctness is shown as follows:

$$\begin{aligned}\prod_i[F_K(z_i)]^{h_i(x)}&=\prod_{i_1+\cdots+i_m\le d}\left(g^{k_0k_1^{i_1}\cdots k_m^{i_m}}\right)^{\binom{d}{i_1,\ldots,i_m}\prod_{j=1}^{m}x_j^{i_j}}\\&=g^{k_0\sum_{i_1+\cdots+i_m\le d}\binom{d}{i_1,\ldots,i_m}\prod_{j=1}^{m}x_j^{i_j}k_j^{i_j}}\qquad(7)\\&=g^{k_0(1+k_1x_1+\cdots+k_mx_m)^d}\end{aligned}$$

where equality (7) follows from the multinomial theorem.

Theorem 4.2. *Let $d \in \mathbb{N}$, and suppose that the d-Strong DDH assumption holds. Then, $\mathcal{PRF}_{2,d}$ is a pseudorandom function.*

Proof. Our proof proceeds in two steps: first we apply the strong DDH assumption to replace, for each $i \in [m]$, $j \in [d]$, k_i^j with a random value $k_{i,j}$. In the second step, we reduce the security of the modified scheme to the security of the Naor-Reingold PRF [51]. Since the latter is proven secure under the DDH Assumption (which is implied by the d-Strong DDH one) our Theorem follows.

Step 1. Consider the following variant $\mathcal{PRF}'_{2,d}$ of $\mathcal{PRF}_{2,d}$:

$\mathsf{KeyGen}'(1^n, m)$**:** The key generation algorithm is modified to output $K = (k_0, K')$ where

$$K' = \begin{pmatrix} k_{1,0} & k_{1,1} & \cdots & k_{1,d} \\ \vdots & \ddots & \vdots & \\ k_{m,0} & k_{m,1} & \cdots & k_{m,d} \end{pmatrix}$$

above $k_{i,0} = 1_{\mathbb{G}}$ for $1 \leq i \leq m$, and $k_{i,j}$ are randomly chosen from $\mathbb{Z}_p$ for all i and $j \geq 1$.

$F'_K(x_1, \ldots, x_m)$**:** Recall that $0 \leq x_i \leq d$ for all i. Compute and output $g^{k_0 \prod_{i=1}^m k_{i,x_i}}$.

We now show that any adversary that can distinguish between an oracle for F_K and an oracle for F'_K can be used to break the d-strong DDH assumption. Suppose that A is such an adversary that can distinguish the two PRF families with advantage $\varepsilon = \varepsilon(n)$, where n is the security parameter. We now construct an adversary B that simulates A, and breaks the d-Strong DDH assumption with probability close to ε.

First, consider the following sequence of hybrid PRFs: $\mathcal{PRF}^i_{2,d}$, for $0 \leq i \leq m$, sets $(k_{j,1}, \ldots, k_{j,d})$ to random values for $1 \leq j \leq i$, and sets $(k_{j,1}, \ldots, k_{j,d}) = (k_j, k_j^2, \ldots, k_j^d)$ for $i < j \leq m$. Then, $\mathcal{PRF}^0_{2,d}$ and $\mathcal{PRF}^m_{2,d}$ are identical to $\mathcal{PRF}_{2,d}$ and $\mathcal{PRF}'_{2,d}$ respectively. Clearly, if A distinguishes between $\mathcal{PRF}_{2,d}$ and $\mathcal{PRF}'_{2,d}$ with advantage ε, then there exists $1 \leq i \leq m$ such that A distinguishes between $\mathcal{PRF}^{i-1}_{2,d}$ and $\mathcal{PRF}^i_{2,d}$ with advantage $\geq \varepsilon/m$.

B works as follows: B is given a sequence of group elements $g, t_1, \ldots, t_d$, where t_j is either g^{x^j} for a random $x \in \mathbb{Z}_p$, or a random group element from $\mathbb{G}$. B then chooses randomly $i \in_{\mathrm{R}} \{1, \ldots, m\}$, and employs the so called "plug-and-pray" technique by implicitly setting $(g^{k_{i,1}}, \ldots, g^{k_{i,d}}) = (t_1, \ldots, t_d)$. B then generates $(k_{j,1}, \ldots, k_{j,d})$ randomly for $1 \leq j < i$, and sets $(k_{j,1}, \ldots, k_{j,d}) = (k_j, k_j^2, \ldots, k_j^d)$ for $i < j \leq m$.

B simulates A. To answer a query $(x_1, \ldots, x_m)$ made by A to the PRF oracle, B sets $y = t_{x_i}^{k_0 \prod_{j \neq i} k_{j,x_j}}$, and returns y. We get that B simulates $\mathcal{PRF}^{i-1}_{2,d}$ if the challenge $(t_1, \ldots, t_d)$ is pseudorandom, and $\mathcal{PRF}^i_{2,d}$ if the challenge is random. B then uses the output of A to distinguish random and pseudorandom tuples. The advantage of B is therefore $\geq \varepsilon/m$ if B guessed i correctly.

Step 2. The second step of the proof is to reduce the security of $\mathcal{PRF}'_{2,d}$ to the security of the Naor-Reingold PRF. Our reduction proceeds as follows. Given an adversary B that distinguishes $\mathcal{PRF}'_{2,d}$ from a random function with advantage $\varepsilon' = \varepsilon'(n)$, we construct an adversary C that distinguishes a Naor-Reingold PRF for inputs of length $m \cdot d$ from a random function. C simulates B, and given a query $(x_1, \ldots, x_m) \in \{0, \ldots, d\}^m$ it creates a new query $\hat{x} = e_{x_1} | \cdots | e_{x_m}$ where e_i is a binary string of length $d + 1$ containing 0 everywhere except at position i, where its value is 1. Above, "|" denotes concatenation. C then submits $\hat{x}$ to its own oracle to obtain a response y, and returns y to B. Finally, C outputs what B outputs.

Recall that the Naor-Reingold PRF requires a key with $m \cdot d$ values in $\mathbb{Z}_p$ to work on binary inputs of length $m \cdot d$. The above construction partitions the elements of the key into m blocks of size d. This in turn yields an identical distribution on the output as the one output by $\mathcal{PRF}'_{2,d}$ for m inputs. □

Remark 4.3. It is interesting to note that the Naor-Reingold PRF is a special case of Construction 2 obtained by setting $d = 1$. Therefore, our construction provides a tradeoff between the security assumption and the size of the key of the PRF: to operate on binary inputs of length n our construction requires $n/\log_2(d+1)$ elements of $\mathbb{Z}_p$ in the key.

Remark 4.4. One can change the above construction so that it becomes slightly less efficient but secure under the *standard DDH* assumption. This modified version is presented in Construction 4 as $\mathcal{PRF}_{4,d}$.

4.2 Small Domain Algebraic PRFs from DDH

Construction 3. Let $\mathcal{G}$ be a computational group scheme. We define $\mathcal{PRF}_3$ as follows:

$\mathsf{KeyGen}(1^n, m)$: Generate a group description $(p, g, \mathbb{G}) \leftarrow_{\mathrm{R}} \mathcal{G}(1^n)$. Choose $k_0, k_{1,1}, \ldots, k_{1,s}, \ldots, k_{m,1}, \ldots, k_{m,s} \in_{\mathrm{R}} \mathbb{Z}_p$. Output $param = ((m, s), p, g, \mathbb{G})$, $K = (k_0, k_{1,1}, \ldots, k_{1,s}, \ldots, k_{m,1}, \ldots, k_{m,s})$.

$F_K(x)$: Interpret $x = (x_1, \ldots, x_m)$ with each $x_i = [x_{i,1}, \ldots, x_{i,s}]$ as an s-bit string. Compute and output $g^{k_0 k_{1,1}^{x_{1,1}} \cdots k_{1,s}^{x_{1,s}} \cdots k_{m,1}^{x_{m,1}} k_{m,s}^{x_{m,s}}}$.

Closed form for polynomials of degree d in each variable. We describe an efficient closed form for $\mathcal{PRF}_3$ for computing polynomials of the form

$$p(x_1, \ldots, x_m) = \sum_{i_1, \ldots, i_m \le d} F_K(i_1, \ldots, i_m) x_1^{i_1} \cdots x_m^{i_m}$$

where the PRF F is initialized with m and $s = \lceil \log d \rceil$. Let $h : \mathbb{Z}_p^m \to \mathbb{Z}_p^l$, where $l = md$, be defined as $h(x_1, \ldots, x_m) = (x_1^{i_1} \cdots x_m^{i_m})_{i_1, \ldots, i_m \le d}$. Let $z = [z_1, \ldots, z_l] = [(i_1, \ldots, i_m)]_{i_1, \ldots, i_m \le d}$ then

$$\mathsf{CFEval}_{h,z}(x_1, \ldots, x_m, K) \stackrel{\mathsf{def}}{=} g^{k_0 \prod_{j=1}^{m} (1 + k_{j,1} x_j)(1 + k_{j,2} x_j^2) \cdots (1 + k_{j,s} x_j^{2^s})}$$

Correctness follows directly by expanding the expression in the exponent.

Remark 4.5. Note that for $m = 1$ we obtain an alternative construction for single-variable polynomials of degree d. Below we prove that Construction 3 is a PRF under the DDH Assumption. Therefore compared to Construction 1, this construction relies on a weaker assumption (DDH vs. D-strong DDH). However the efficiency of the closed form computation in Construction 1 is better: constant vs. $O(\log d)$ in Construction 3. Jumping ahead this will give us two alternative ways to delegate the computation of a single-variable polynomial of degree d with the following tradeoff: either one assumes a weaker assumption (DDH) but verification of the result will take $O(\log d)$ time, or one assumes a stronger assumption to obtain constant verification time.

Closed form for 1-out-of-2 multivariate polynomials of degree 1. We now consider polynomials of the form

$$p(x_1, y_1, \ldots, x_m, y_m) = \sum_{s \in \{0,1\}^m} F_K(s) x_1^{s_1} y_1^{1-s_1} \cdots x_m^{s_m} y_m^{1-s_m}$$

In such polynomials, each monomial contains exactly one of x_i and y_i for $1 \le i \le m$. We initialize the PRF F with m and $s = 1$ (and for simplicity we drop the double subscript and denote the key

$k_{i,1}$ as k_i). Specifically, let $h : \mathbb{Z}_p^{2m} \to \mathbb{Z}_p^l$, where $l = 2^m$, be defined as $h(x_1, y_1, \ldots, x_m, y_m) = (x_1^{s_1} y_1^{1-s_1} \cdots x_m^{s_m} y_m^{1-s_m})_{s \in \{0,1\}^m}$. We can then define

$$\mathsf{CFEval}_h(x_1, y_1, \ldots, x_m, y_m) \stackrel{\mathsf{def}}{=} g^{k_0(x_1 + k_1 y_1) \cdots (x_m + k_m y_m)}$$

Correctness is straightforward by expanding the expression in the exponent. The proof of the following theorem was given in [51];

Theorem 4.6. *[51] Suppose that the DDH assumption holds for $\mathcal{G}$. Then, $\mathcal{PRF}_3$ is a pseudorandom function.*

Construction 4. Let $\mathcal{G}$ be a computational group scheme. We define $\mathcal{PRF}_{4,d}$, for $d \in \mathbb{N}$, as follows:

$\mathsf{KeyGen}(1^n, m)$: Generate a group description $(p, g, \mathbb{G}) \leftarrow_{\mathrm{R}} \mathcal{G}(1^n)$. Choose $k_{1,0}, \ldots, k_{1,m}, \ldots, k_{d,0}, \ldots, k_{d,m} \in_{\mathrm{R}} \mathbb{Z}_p$. Output $param = (m, p, g, \mathbb{G})$, $K = (k_{1,0}, \ldots, k_{d,m})$.

$F_K(x)$: Interpret x as a vector $(x_1, \ldots, x_d) \in \{0, \ldots, m\}^d$. Compute and output $g^{k_0 \prod_{j=1}^d k_{j,x_j}}$.

Closed form for m-variate polynomials of total degree at most d. We describe an efficient closed form for $\mathcal{PRF}_{4,d}$ for computing polynomials of the form

$$p(x_1, \ldots, x_m) = \sum_{\substack{i_1, \ldots, i_m \\ i_1 + \cdots + i_m \leq d}} F_K(i_1, \ldots, i_m) x_1^{i_1} x_2^{i_2} \cdots x_m^{i_m}.$$

Let $h : \mathbb{Z}_p^m \to \mathbb{Z}_p^l$, where $l = (m+1)^d$, be defined as

$$h(x_1, \ldots, x_m) \stackrel{\mathsf{def}}{=} \left(\prod_{\substack{j=1 \\ i_j \neq 0}}^{d} x_{i_j} \right)_{i_1, \ldots, i_d \in \{0, \ldots, m\}}$$

Let $z = [z_1, \ldots, z_l] = [(i_1, \ldots, i_d)]_{i_1, \ldots, i_d \in \{0, \ldots, m\}} \in \mathbb{Z}_m^{d \times l}$. We can now define

$$\mathsf{CFEval}_{h,z}(x_1, \ldots, x_m, K) \stackrel{\mathsf{def}}{=} g^{\prod_{i=1}^d (k_{i,0} + k_{i,1} x_1 + \cdots + k_{i,m} x_m)}$$

Correctness follows by algebraic manipulation and is given in the full version of this paper.

Theorem 4.7. *Let $d \in \mathbb{N}$, and suppose that the DDH assumption holds. Then, $\mathcal{PRF}_{4,d}$ is a pseudorandom function.*

The proof is a small adaptation of that of Theorem 4.2. The main observation is that the $k_{i,j}$ values are never raised to a power more than one so the output of the PRF is in fact a subset of the bits of the Naor-Reingold PRF [51].

Remark 4.8. It is interesting to note that constructions 2 and 4 solve the exact same problem. While the former has better parameters, the later works under a weaker assumption.

Polynomial Type	Setup	C. Query	S. Query	Assumption	$\mathcal{PRF}$
1-variable, degree d	$O(d)$	$O(1)^4$ (1)	$O(d)$	d-Strong DDH	$\mathcal{PRF}_1$
n-variable, variable degree d	$O((d+1)^n)$	$O(n \log d)$ (1)	$O((d+1)^n)$	DDH	$\mathcal{PRF}_3$
n-variable, total degree d	$O(\binom{n+d}{d})$	$O(n \log d)$ (1)	$O(\binom{n+d}{d})$	d-Strong DDH	$\mathcal{PRF}_{2,d}$
n-variable, total degree d	$O((n+1)^d)$	$O(nd)$ (1)	$O((n+1)^d)$	DDH	$\mathcal{PRF}_{4,d}$

Table 1: Parameters of different protocols for verifiable delegation of polynomials. Numbers inside the parenthesis show the number of group operations. Columns starting with "C." are the client's requirements and the ones starting with "S." are the server's. In each case the server's query runtime (resp. space requirements) is asymptotically the same as evaluating (resp. storing) the polynomial. Note that $(\frac{n+1}{\sqrt{d+1}})^d \leq \binom{n+d}{d} \leq (n+d)^d$, and in particular for constant d it is $\Theta(n^d)$.

5 Verifiable Delegation of Polynomials

The basic idea of our construction is the following. The client stores the polynomial $P(\cdot)$ in the clear with the server as a vector $\boldsymbol{c}$ of coefficient in $\mathbb{Z}_p$. The client also stores with the server a vector $\boldsymbol{t}$ of group elements of the form $g^{ac_i+r_i}$ where $a \in_R \mathbb{Z}_p$ and r_i is the i^{th}-coefficient of a polynomial $R(\cdot)$ of the same degree as $P(\cdot)$. When queried on input x the server returns $y = P(x)$ and $t = g^{aP(x)+R(x)}$ and the client accepts y iff $t = g^{ay+R(x)}$.

If $R(\cdot)$ was a random polynomial, then our proof below shows that this is a secure delegation scheme. However checking that $t = g^{ay+R(x)}$ would require the client to evaluate the random polynomial $R(\cdot)$, which is just as inefficient as evaluating the original polynomial $P(\cdot)$. Moreover, the client would have to store a long description of a random polynomial that is as long as the original polynomial[3] $P(\cdot)$. The crucial point, therefore, is how to make this computation fast. We do that by defining $r_i = F_K(i)$ for an algebraic PRF that has a closed form efficient computation for polynomials, such as the ones described in the previous section. Since F is a PRF, the security of the scheme is not compromised, and the closed form efficiency of F will allow the client to verify the result in time sub-linear in the degree of the polynomial.

The result is described in general form, using algebraic PRFs. It therefore follows that we obtain efficient and secure delegation protocols not only for single-variable polynomials of degree d, but also for multivariate polynomials with total degree d or of degree d in each variable. The relevant parameters of the resulting protocols for each of these cases can be seen in Table 1.

Finally at the end of the section we show how to protect the privacy of the polynomial, by encrypting it with an homomorphically additive encryption scheme.

5.1 Construction Based on Algebraic PRFs

We describe a verifiable delegation scheme for functions of the form $f_{\boldsymbol{c},h}(\boldsymbol{x}) = \langle h(\boldsymbol{x}), \boldsymbol{c} \rangle$, where $\boldsymbol{c}$ is a (long) vector of coefficients, $\boldsymbol{x}$ is a (short) vector of inputs, and h expands $\boldsymbol{x}$ to a vector of the same length of $\boldsymbol{c}$. Our construction is generic based on any algebraic PRF that has closed form efficiency relative to h.

Protocol Delegate-Polynomial($\boldsymbol{c}$)

KeyGen$(\boldsymbol{c}, n)$**:** Generate $(K, param) \leftarrow_R \mathsf{KeyGen}(1^n, \lceil \log d \rceil)$. Parse $\boldsymbol{c}$ as a vector $\boldsymbol{c} = (c_0, \ldots, c_d) \in \mathbb{Z}_p^{d+1}$. Let $\mathbb{G}$ be the range group of F_K, and let g be a generator for that group. Compute $g_i \leftarrow F_K(i)$

[3] Alternatively, the client could generate the coefficients of R using a standard PRF, thereby avoiding storing a large polynomial. However, this would still require the client to recompute all the coefficients of R each time a verification needs to be performed

[4] The client needs to do two exponentiations.

for $0 \leq i \leq d$, choose $a \in_R \mathbb{Z}_p$, and set $\boldsymbol{t} = [t_0, \ldots, t_d] \leftarrow (g_0 g^{ac_0}, \ldots, g_d g^{ac_d})$. Output $PK \leftarrow (param, \boldsymbol{c}, \boldsymbol{t})$, and $SK \leftarrow (K, a)$.

$\mathbf{ProbGen}(SK, \boldsymbol{x})$: Output $(\sigma_x, \tau_x) = (\boldsymbol{x}, \boldsymbol{x})$.

$\mathbf{Compute}(PK, \sigma_x)$: Parse PK as $(param, \boldsymbol{c}, \boldsymbol{t})$, $\boldsymbol{c}$ as $c_0, \ldots, c_d$, and σ_x as $\boldsymbol{x}$. Compute $\boldsymbol{w} \leftarrow h(\boldsymbol{x}) = [h_0(\boldsymbol{x}), \ldots, h_d(\boldsymbol{x})] \in \mathbb{Z}_p^{d+1}$, $y \leftarrow \sum_{i=0}^{d} c_i h_i(\boldsymbol{x})$, and $t \leftarrow \prod_{i=0}^{d} t_i^{h_i(\boldsymbol{x})}$. Output $\nu_x = (y, t)$.

$\mathbf{Verify}(SK, \tau_x, \nu_x)$: Parse SK as (K, a), τ_x as $\boldsymbol{x}$, and ν_x as $(y, t) \in \mathbb{Z}_p \times \mathbb{G}$. Compute $z \leftarrow \mathsf{CFEval}_h(\boldsymbol{x}, K)$, and accept if $t \stackrel{?}{=} z \cdot g^{a \cdot y}$. Otherwise, reject.

CORRECTNESS. The correctness of the above scheme follows straightforwardly from the correctness of CFEval for the algebraic PRF F.

5.2 Analysis

We now prove that our verifiable delegation scheme is secure under the assumption that (KeyGen, F) is a family of pseudorandom functions.

Theorem 5.1. *Suppose that* (KeyGen, F) *is a secure pseudorandom function scheme. Then,* Delegate-Polynomial *is a secure verifiable delegation scheme according to Definition 3.2.*

Our proof proceeds in two steps: first, we switch to an information theoretic setting, where F_K is replaced by a random function. Then, we apply an information theoretic argument to obtain security. Formally, the proof proceeds by a sequence of games. For each game i we denote by X_i the event that the adversary wins (by successfully convincing the verifier to accept a false output).

Game 0. Game 0 is the original security experiment, played with the adversary A.

Game 1. Game 1 is the same as Game 0, except that verification is done differently: instead of using CFEval, the challenger now computes z directly by using the algebraic properties of the PRF: let $\boldsymbol{g} = (g_0 = F_K(0), \ldots, g_d = F_K(d))$, then, $z = \prod_i g_i^{h(\boldsymbol{x})_i}$.

Game 2. Game 2 is the same as Game 1, except that the group elements $g_0, \ldots, g_d$ are chosen randomly from the range $\mathbb{G}$ of the PRF.

Claim 5.2. $\Pr[X_0] = \Pr[X_1]$

Proof. The difference between Games 0 and 1 is only in the way that the value z is computed during verification. In Game 1 it is computed directly using the algebraic properties of the PRF, while in Game 0, the key K of the PRF is used to perform the computation efficiently. In both cases, $z = \prod_i g_i^{h(\boldsymbol{x})_i}$.Therefore, the change is purely conceptual, and the probability that the adversary wins is not affected. □

Claim 5.3. $|\Pr[X_1] - \Pr[X_2]| \leq \varepsilon_{\mathsf{PRF}}$

Proof. The proof is by straightforward reduction to the security of the pseudorandom function (KeyGen, F). □

Before proceeding to prove that the probability of winning Game 2 is small we state and prove a Lemma which will be useful in the proof. Consider the following experiment, (all $\equiv$ signs show congruence modulo n)

Experiment $\mathbf{Exp}_A[\mathbb{Z}_n, m]$
$a \in \mathbb{Z}_n$ is chosen uniformly at random;
For $i = 1, \ldots, m$;
$x_i \leftarrow A(n, m, x_1, y_1, \ldots, x_{i-1}, y_{i-1})$;
$r_i \in \mathbb{Z}_n$ is chosen uniformly at random;
$y_i \leftarrow ax_i + r_i$;
$A^{\mathcal{O}_{a,(r_i)_i,(x_i)_i}(\cdot,\cdot,\cdot)}(x_1, \ldots, x_m, y_1, \ldots, y_m, n)$ is ran with the following oracle:
$\mathcal{O}_{a,(r_i)_i,(x_i)_i}(y, x, (c_i)_i) = \begin{cases} 1 & \text{if } y \equiv ax + \sum_i c_i r_i \text{ and } x \not\equiv \sum_i c_i x_i, \\ 0 & \text{otherwise} \end{cases}$;
If $\mathcal{O}$ ever output 1 to A output 1, otherwise output 0;

Lemma 5.4. *If a (computationally unbounded) adversary A always makes at most q queries to $\mathcal{O}$ in the above experiment its success probability is at most q/p where p is the smallest prime factor of n.*

Proof. First notice that we can make the interaction of A and $\mathcal{O}$ completely non-adaptive. In other words, if A ever gets a 1 from $\mathcal{O}$ it has already won the game so once A knows $x_1, \ldots, x_m, y_1, \ldots, y_m$ and its randomness it is going to make the same series of queries to $\mathcal{O}$, except perhaps stopping when he gets 1 as an answer. We can then change the last two steps of the experiment as follows,

$((y'_1, x'_1, (c_i^1)_i), \ldots, (y'_q, x'_q, (c_i^q)_i)) \leftarrow A(x_1, \ldots, x_m, y_1, \ldots, y_m, n)$;
If there is a $1 \le j \le q$ such that $y'_j \equiv ax'_j + \sum_i c_i^j r_i$ and $x'_j \not\equiv \sum_i c_i^j x_i$ output 1, otherwise output 0;

We can now bound the probability that the result of this experiment is 1 as follows. Without loss of generality we assume that the adversary never outputs $(y'_j, x'_j, (c_i^j)_i)$ such that $x'_j \equiv \sum_i c_i^j x_i$ as such outputs can be avoided and do not help the adversary. We have,

$$\Pr[\text{output is 1}] = \Pr\left[\bigvee_j y'_j \equiv ax'_j + \sum_i c_i^j r_i\right] \le \sum_j \Pr\left[y'_j \equiv ax'_j + \sum_i c_i^j r_i\right].$$

We will now consider the probability inside the sum conditioned on a particular value for $x_1, \ldots, x_m$, $y_1, \ldots, y_m$ showing that no matter what they are it is at most $1/p$. First notice that as the adversary is computationally unbounded, without loss of generality one can assume that it is deterministic. Now fix such a value for x_i's and y_i's that happens in the experiment with non-zero probability. Notice that once we fix the value of a, no matter what it is, there is a unique value for r_1 that results in the answer y_1 for the adversary's first query, x_1. There in then one unique value of r_2 that makes the answer to the query x_2, y_2, and so on and so forth. In other words there are exactly n possible values of $a, r_1, \ldots, r_m$ that result in these queries and answers, precisely one for each possible choice of a. Now if $y'_j \equiv ax'_j + \sum_i c_i^j r_i$, we have,

$$y'_j - \sum_i y_i c_i \equiv ax'_j + \sum_i c_i^j r_i - a\left(\sum_i c_i x_i\right) - \sum_i c_i^j r_i \equiv a\left(x'_j - \sum_i c_i x_i\right).$$

Given that $x'_j - \sum_i c_i^j x_i \not\equiv 0 (\text{mod } n)$ the same quantity is also nonzero modulo some prime factor of n, say $p_0 | n$. In other words, $a \equiv \left(x'_j - \sum_i c_i^j x_i\right)^{-1}\left(y'_j - \sum_i y_i c_i\right) (\text{mod } p_0)$. But there are exactly n/p_0 such

a. In other words, out of the n possible settings of $a, r_1, \ldots, r_m$ at most n/p_0 will result in the adversary succeeding.

Putting everything together we can write,

$$\begin{aligned}\Pr\left[\text{output is 1}\right] &\leq \sum_j \Pr\left[y'_j \equiv ax'_j + \sum_i c_i^j r_i\right] \\ &= \sum_j \mathbb{E}_{x_1\ldots,x_m,y_1,\ldots,y_m} \Pr\left[y'_j \equiv ax'_j + \sum_i c_i^j r_i \middle| x_1 \ldots, x_m, y_1, \ldots, y_m\right] \leq q/p_0 \leq q/p.\end{aligned}$$

Note that the distribution on x_i's and y_i's in the second line is not uniform but rather depends on how the adversary computes its queries. However, the bounds on the conditional probabilities hold for any value of these variables.

□

We will make use of the following corollary in the analysis of our updatable verifiable database scheme in Section 6.2:

Corollary 5.5. *Lemma 5.4 holds even when the adversary submits pairs (x_i, x'_i) of elements in $\mathbb{Z}_N$ and sees $ax_i + bx'_i + r_i$, where $b \in_R \mathbb{Z}_N$.*

Claim 5.6. $\Pr[X_2] \leq l/p$*, where l is the number of the queries the adversary makes and p is the size of the group.*

Proof. We show that in Game 2 the advantage of the adversary in winning the game is negligible unconditionally. Let us denote $g_i = g^{r_i}$ for $1 \leq i \leq d$, where $r_i \in_R \mathbb{Z}_p$ is a random exponent. Then, $\boldsymbol{t} = (t_0, \ldots, t_d)$ where $t_i = g^{r_i + ac_i}$. Let $(x_0, \ldots, x_d) \leftarrow h(\boldsymbol{x})$. Our goal is to show that with all but negligible probability, the client will only accept the correct value $y = c_0x_0 + \cdots + c_dx_d$.

Let us review the check performed by the client. First, the client computes $z = g_0^{x_0} \ldots g_d^{x_d} = g^{r_0x_0 + \cdots + r_dx_d}$. Then, to check a response (y', t) produced by the server, the client checks if $t \stackrel{?}{=} zg^{ay'} = g^{\sum r_ix_i + ay'}$.

We will use Lemma 5.4 with parameters $n = p$ and m equal to the dimension of the vector c. Assume that such an adversary, A, exists. We will show how to construct an adversary A' for Lemma 5.4. In the first step of the experiment of Lemma 5.4 a random a is chosen. In the second step A' will output $x_1 = c_1, x_2 = c_2, \ldots$ and receive $y_i = r_i + ac_i$. It will compute $t_i = g^{y_i}$ and submit $\mathbf{t} = (t_1, \ldots, t_d), \mathbf{c} = (c_1, \ldots, c_d)$ to A. Each time A outputs an x_i to be given to $\mathbf{ProbGen}$ we run $\mathbf{ProbGen}(x_i)$, and when A outputs a $\sigma_{y,i} = (y_i, t_i)$ we run the oracle $\mathcal{O}$ on $(\log_g t_i, y_i, h(\mathbf{x}))$, where $\log_g t_i$ is the discrete log of t_i, the number for which $g^{\log_g t_i} = t_i$.[5] Notice that as Lemma 5.4 applies to any adversary (regardless of runtime) we can find the discrete log of t_i. They are several cases depending on the output of $\mathbf{Verify}$. The $\mathbf{Verify}$ step will result in $f(x)$ if and only if $\langle h(\mathbf{x}), \mathbf{c}\rangle = y_i$ which is the case where $\mathcal{O}$ answers 0 because of the second condition. It will result in $\perp$ if and only $\mathcal{O}$ outputs 0, because of the first condition. And finally, $\mathbf{Verify}$ outputs something other than $f(x)$ and $\perp$ if and only if $\mathcal{O}$ outputs 1.

To sum everything up the success probability of A' is exactly the same as that of A. □

5.3 Encrypting The Polynomial

The solution described above has the drawback that the client must store the polynomial *in the clear* with the server. This problem can be solved by encrypting the coefficients of the polynomial with an additively-homomorphic encryption scheme. In this case, after the client provides the value x, the server can compute

[5] notice that y_i is used as the x of Lemma 5.4.

an encryption of $P(x)$ and return it to the client who will decrypt it. The verification step remains the same. Details follow.

Let $\mathcal{E} =$AH-KG, AH-Enc, AH-Dec, denote a public-key probabilistic encryption scheme. Let the message space be a group under addition and the ciphertext space a group under multiplication. We say that $\mathcal{E}$ is a additively-homomorphic if for any public key pk of the encryption scheme, given $\gamma_1 = \mathsf{AH} - \mathsf{Enc}_{pk}(m_1; r_1)$ and $\gamma_2 = \mathsf{AH} - \mathsf{Enc}_{pk}(m_2; r_2)$, there exists an r such that

$$\gamma_1 \cdot \gamma_2 = \mathsf{AH} - \mathsf{Enc}_{pk}(m_1 + m_2; r).$$

Note that if one encrypts the coefficient $(c_0, \ldots, c_d)$ of a polynomial $P(\cdot)$, $\gamma_i = \mathsf{AH} - \mathsf{Enc}_{pk}(c_i)$ then for any value x one can compute $\mathsf{AH} - \mathsf{Enc}_{pk}(P(x)) = \prod_{i=0} \gamma_i^{x^i}$. An example of a good additively homomorphic scheme for our purposes is [54].

Protocol Private-Delegate-Polynomial($\boldsymbol{c}$)

KeyGen$(\boldsymbol{c}, n)$**:** Generate $(K, param) \leftarrow_{\mathrm{R}} \mathsf{KeyGen}(1^n, \lceil \log d \rceil)$. Parse $\boldsymbol{c}$ as a vector $\boldsymbol{c} = (c_0, \ldots, c_d) \in \mathbb{Z}_p^{d+1}$. Let $\mathbb{G}$ be the range group of F_K, and let g be a generator for that group. Compute $g_i \leftarrow F_K(i)$ for $0 \leq i \leq d$, choose $a \in_{\mathrm{R}} \mathbb{Z}_p$, and set $\boldsymbol{t} = [t_0, \ldots, t_d] \leftarrow (g_0 g^{ac_0}, \ldots, g_d g^{ac_d})$.

Generate $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{AH} - \mathsf{KG}(1^n)$, a public/private key pair for an additively homomorphic encryption scheme. Compute $\gamma_i = \mathsf{AH} - \mathsf{Enc}_{pk}(c_i)$

Output $PK \leftarrow (param, pk, \boldsymbol{\gamma}, \boldsymbol{t})$, and $SK \leftarrow (K, sk, a)$.

ProbGen$(\boldsymbol{SK}, \boldsymbol{x})$**:** Output $(\sigma_x, \tau_x) = (\boldsymbol{x}, \boldsymbol{x})$.

Compute$(\boldsymbol{PK}, \sigma_x)$**:** Parse PK as $(param, pk, \boldsymbol{\gamma}, \boldsymbol{t})$, $\boldsymbol{\gamma}$ as $\gamma_0, \ldots, \gamma_d$, and σ_x as $\boldsymbol{x}$. Compute $\boldsymbol{w} \leftarrow h(\boldsymbol{x}) = [h_0(\boldsymbol{x}), \ldots, h_d(\boldsymbol{x})] \in \mathbb{Z}_p^{d+1}$, $u \leftarrow \prod_{i=0}^{d} \gamma_i^{h_i(\boldsymbol{x})}$, and $t \leftarrow \prod_{i=0}^{d} t_i^{h_i(\boldsymbol{x})}$. Output $\nu_x = (z, t)$.

Verify$(\boldsymbol{SK}, \tau_x, \nu_x)$**:** Parse SK as (K, sk, a), τ_x as $\boldsymbol{x}$, and ν_x as (u, t). Compute $y = \mathsf{AH} - \mathsf{Dec}_{sk}(u)$ Compute $z \leftarrow \mathsf{CFEval}_h(\boldsymbol{x}, K)$, and accept if $t \stackrel{?}{=} z \cdot g^{a \cdot y}$. Otherwise, reject.

Notice that although the privacy of the polynomial is preserved, the client still reveals to the server the points on which he wants the polynomial evaluated.

Theorem 5.7. *Suppose that* (KeyGen, F) *is a secure pseudorandom function scheme, and* $\mathcal{E} =$AH-KG, AH-Enc, AH-Dec *a semantically secure additively homomorphic encryption scheme. Then,* Private-Delegate-Polynomial *is a secure verifiable delegation scheme according to Definition 3.2. Moreover for any two* d*-degree polynomials* P_0, P_1*, the views of the server in* Private-Delegate-Polynomial *when the client uses either* P_0 *or* P_1 *are computationally indistinguishable.*

The proof is an easy extension of the proof of Theorem 5.1 and will be presented in the final version.

6 Verifiable Database Queries with Efficient Updates

We have shown a general framework that allows any resource constrained client to verifiably delegate large polynomials to a server. As we have already mentioned in the introduction, this immediately gives a verifiable delegation solution to many natural practical applications (such as prediction using fitted polynomials). In this Section we present an application of our techniques to the problem of efficient verification of the result to queries posed to a dynamic database. In other words the client stores a database with the server together with some authentication informatiom. It then needs to be able to efficiently verify that the results

of its queries are correct, and also to efficiently update the database and its associated authenticator. The techniques we developed for delegation of polynomials are at the basis of the solution we present, which however requires other novel and interesting technical ideas.

We describe our solution in steps. First we show a simple solution for the simpler problem of verifiable keyword search, and then we extend it to the case of updatable databases. We also have a generic (but less efficient) way to add updates to any implementation of verifiable keyword search which due to lack of space is presented in Appendix A of the appendix.

6.1 Verifiable Keyword Search

Consider a large text file $F = \{w_0, \ldots, w_d\}$ where w_i are the words appearing in F. Consider the application in which the client stores the file with the server and later issues keyword seach queries: for a given keyword w the server must efficiently prove to the client if $w \in F$ or not.

A SIMPLE SOLUTION BASED ON MACS. As a leading example towards our construction we briefly describe a simple and very efficient construction for this problem based on Message Authentication Codes[6]. The main idea is the following: for each $w \in F$ the client computes $\sigma_w = \mathsf{MAC}_K(w)$, and uploads σ_w to the server. The more interesting case is what to do for all $w \notin F$. Indeed, the trivial solution of allowing the server to return $\perp$ allows the server to falsely claim that a certain element $w \in F$ is, in fact, not in the file (a malicious server can just return $\perp$ whenever it wants to convince a client that something is not in F). The solution is for the client to MAC every shortest prefix u such that no continuation of u is in F (that is, for all $u' \in \{0,1\}^*$, $uu' \notin F$). Fortunately, the number of such prefixes is polynomial, and can be found by constructing a binary tree that is pruned at all the subtrees that contain no elements of F (as is described in [46]). For every such prefix u, the client MACs the special string "$u\perp$". Then, for the client to be convinced that $w \notin X$, the server must produce a MAC of "$u\perp$", where u is a prefix of w.

A SOLUTION BASED ON POLYNOMIAL DELEGATION. Our solution based on polynomial delegation is more expensive than the MAC-based one described above. Yet we present it because (i) has additional properties not enjoyed by the MAC-based solution and (ii) forms the conceptual basis of a scheme in which the client can efficiently update the stored file (described later in this Section). The construction is almost immediate: the client encodes F as the polynomial $P(\cdot)$ of degree d such that $P(w_i) = 0$, and then the client and the server run our verifiable delegation scheme over P (the client accepts $w \in F$ if the server returns $P(w) = 0$). A variation of this protocol yields privacy of the polynomial and of the query w for free, without using any form of encryption. We obtain this through two separate ideas:

1. The client chooses a key κ for a "traditional" PRF family $\{\phi_\kappa\}$. Computes $\chi_i = \phi_\kappa(w_i)$ and defines $P(\cdot)$ as the polynomial of degree d such that $P(\chi_i) = 0$. When querying a keyword w, the client submits $\chi = \phi_\kappa(w)$.

2. The client does not store the polynomial with the server, but just the authentication information since that alone will allow him to determine if $P(\chi) = 0$ or not.

We now describe the complete protocol:

Protocol Polynomial-Keyword-Search($\boldsymbol{w}$)

KeyGen$(\boldsymbol{w}, n)$**:** Generate $(K, param) \leftarrow_{\mathrm{R}} \mathsf{KeyGen}(1^n, \lceil \log d \rceil)$ the keys for an algebraic PRF family $\{F_K\}$ which has a closed form efficiency for d-degree single-variable polynomials (Constructions 1

[6] The main idea is an adaptation to the private key setting of constructions from the area of Zero Knowledge Sets [46] We do not know if this idea has appeared elsewhere before.

or 3). Also generate κ the key for a traditional PRF family $\{\phi_\kappa\}$. Compute $\chi_i = \phi_\kappa(w_i)$ and the polynomial $P(\cdot)$ of degree d such that $P(\chi_i) = 0$. Let $\boldsymbol{c}$ be the vector of its coefficients in $\mathbb{Z}_p$. Let $\mathbb{G}$ be the range group of F_K, and let g be a generator for that group. Compute $g_i \leftarrow F_K(i)$ for $0 \leq i \leq d$, choose $a \in_R \mathbb{Z}_p$, and set $\boldsymbol{t} = [t_0, \ldots, t_d] \leftarrow (g_0 g^{ac_0}, \ldots, g_d g^{ac_d})$. Output $PK \leftarrow (param, \boldsymbol{t})$, and $SK \leftarrow (K, a, \kappa)$.

ProbGen(SK, w): Output $(\sigma_w, \tau_w) = (\chi = \phi_\kappa(w), w)$.

Compute(PK, σ_w): Parse PK as $\boldsymbol{t} = [t_0, \ldots, t_d]$, and σ_w as χ. Compute $t \leftarrow \prod_{i=0}^{d} t_i^{\chi^i}$. Output $\nu_w = t$.

Verify(SK, τ_x, ν_x): Parse SK as (K, a, κ), $\tau_w = w$, and $\nu_x = t$. Compute $\chi = \phi_\kappa(w)$ and $z \leftarrow \mathsf{CFEval}_h(\chi, K) = g^{\sum_{i=0}^{d} F_K(i)\chi^i}$ efficiently using the closed form efficiency of F_K. Accept if, and accept if $t/z = 1$. Otherwise, reject.

ADVANTAGES. While the polynomial-based solution is computationally more expensive than the MAC based solution it has two additional properties not enjoyed by the latter. construction that is described later on in this section:

1. *Privacy of query response against the server.* In the polymonial-based solution the server has no idea from the transcript of the protocol if the client accepted or not (it is not clear how to achieve this in the MAC-based solution).

2. *Deniability.* The client can always *deny* the content of the file to a third-party who knows the information stored at the server. Indeed the client can always come up with a "fake" private key $\hat{K}, \hat{\kappa}, \hat{a}$ (choosing them uniformly at random) which implicitly defines a polynomial $\hat{P}$ that matches the vector $\boldsymbol{t}$ stored at the server[7].

VERIFIABLE DELEGATION OF A DATABASE. Both solutions above can be extended to the verifiable delegation of database queries. Consider a database of the form $\{(x_1, v_1), \ldots, (x_d, v_d)\}$ where x_i is the *index* and v_i is the *payload* (data) associated with x_i.

The extension is simple. Run the keyword search scheme on the indices, and then when an index is in the database, also return the payload with a MAC on it. We note that in this case the deniabilty and privacy advantages of our polynomial scheme over the tree-based scheme do not hold anymore (since the server stores the MACs of all the values in the database).

6.2 A Verifiable Database Protocol with Efficient Updates

We are now finally ready to present the protocol for verifiable database queries with efficient updates.

The protocol uses ideas borrowed from our polynomial verification scheme: the authenticator for every database entry can be efficiently reconstructed by the client using a PRF with closed form efficiency. However as we pointed out in the Introduction the main challenge comes with the updates: the client must revoke any authenticating data that the server has for the previous value of the updated cell. We deal with this problem by "masking" the authenticator with another closed-form efficient PRF. This "mask" can be efficiently removed by the client to perform the authentication and can also be efficiently updated so that old masked values cannot be reused. The technical details are somewhat involved and are described below.

HANDLING LARGE PAYLOADS. For simplicity we consider databases of the form (i, v_i) where i is the index and v_i the payload data. The construction we describe below allows data values to be only polynomially large

[7] The client can actually choose $\hat{a}$ appropriately after selecting $\hat{K}, \hat{\kappa}$ so that the polynomial $\hat{P}$ satisfy one condition $\hat{P}(x) = y$.

(in the security parameter). Before proceeding to describe our protocol, we show a simple transformation that allows us to support databases with arbitrary payload sizes.

On a high level, we will use a small payload protocol, such as the one described in Construction 6.4, to store a database of the form (i, s_i) where s_i is a counter that counts the number of times index i has been updated. The server will also store the MAC of the tuple (i, s_i, v_i) where v_i is the (possibly large) payload.

When the client queries i, it will receive the value s_i through the verifiable database protocol. The security of this protocol will guarantee to the client that s_i is correct. Then the server will also answer with v_i and the MAC on (i, s_i, v_i) and the client will accept v_i if the MAC is correct. To update index i, the client will first query i and retrieve the current tuple (i, s_i, v_i). It will then update s_i on the verifiable database by setting $s'_i = s_i + 1$. This can be done since s_i is bounded by the running time of the client and therefore polynomial. Finally it will store the new v'_i together with a MAC on (i, s'_i, v'_i). Since s_i will only ever increase, the server will not be able to re-use old MACs once an index has been updated.

Therefore for now we will focus on databases with small (polynomial) payloads. The protocol is described in detail below.

RELATION TO MERKLE TREES. Merkles trees [44] are a standard technique for efficient authentication of data. Each element is represented as a leaf of a binary tree. The internal nodes contain hashes of their two children, and the owner of the data keeps only the root, which is essentially a hash of the entire tree. Retrieval queries can now be answered and verified in logarithmic time: the server simply sends the hashes along the path from the root to the desired leaf (along with any hashes within distance 1 of the path), and the client uses these values to compute the hash at the root of the tree. The client then checks that the stored hash value of the root is equal to the recomputed hash. Updating the tree is also quite efficient – only the hashes along the path to the updated leaf must be recomputed. In comparison, our scheme requires the client to perform a constant amount of work both during retrieval and updates, while the server must choose one of the two types of queries where he will do a linear amount of work (the other type of queries requires a constant amount of work from the server as well).

OUR PROTOCOL. The basic tools that we use are computational bilinear groups of composite order. In this setting, a pair of groups $\mathbb{G}, \mathbb{G}_T$ are generated, along with a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Here each of the groups are of some order $N = p_1 p_2$ where p_1 and p_2 are large primes. The cryptographic assumption is that it is infeasible to distinguish a random member of $\mathbb{G}$ (or $\mathbb{G}_T$) from a random member of the subgroup of order p_1 or p_2. Such groups have recently been used to solve several important open problems in the areas of identity based encryption, leakage resilient cryptography, and other related problems (see e.g. [13]).

The basic approach of our construction (leaving some details to the full description below) can be described as follows: each entry (i, v_i) in the database (where i is an index and v_i is data) is encoded as a composite group element of the form

$$t_i = g_1^{r_i + a v_i} g_2^{w_i}.$$

Here, g_1 and g_2 are generators of the subgroups $\mathbb{G}_1$ and $\mathbb{G}_2$ of $\mathbb{G}$, and the values r_i, w_i are generated using a pseudo-random function. To retrieve the value of the database at index i, we will have the server compute (given keys that we shall describe in a moment) the value

$$t = g_1^{r_i + a v_i} g_2^{\sum_i w_i}.$$

Forgetting (for now) how the server should compute these values, the client can easily strip off the $\mathbb{G}_2$ masking by keeping the single group element $g_2^{\sum w_i}$ in his private key. It is now easy to see that if we replace the r_i's with random values, then our scheme is secure before any updates are performed. This follows from the fact that each entry in the database is MAC'ed with an information theoretically secure MAC (the $\mathbb{G}_2$ part hasn't played a role so far), and so the server must return the correct value in the $\mathbb{G}_1$ part

of each entry. The difficulty is in allowing updates that do not require the client to change his keys for the pseudo-random functions, which in turn would require the server to obtain new MACs for all the entries in the database.

A naive solution to change the value of index i from v_i to v_i' can be for the client to send to the server a new encoding $g_1^{r_i+bv_i} g_2^{w_i}$. However, the server can then easily recover the MAC keys r_i and a by dividing the new group element that he receives during the update by the previous encoding that he already has. Our solution is therefore to randomize the new encoding by having the client send

$$t_x' = g_1^{r_i+a\delta} g_2^{w_i'},$$

where $\delta = v_i' - v_i$, and w_i' is a new pseudorandom value (generated by using a counter). Intuitively, this allows the client to send t_x' as an update token that the server can multiply into his existing group element t_i to obtain $g_1^{r_i+av_i'} g_2^{w_i+w_i'}$. Notice that the $\mathbb{G}_1$ part is a MAC of the value v_i' using the same key that was previously used to MAC v_i. We show, relying on the subgroup membership assumption, that the random mask $g_2^{w_i+w_i'}$ effectively makes the MAC in the $\mathbb{G}_1$ of the token indistinguishable from a new MAC using fresh keys.

We now arrive at the problem of allowing the server to compute the value t, which requires stripping the $\mathbb{G}_1$ part of all the tokens except the token that corresponds to index i, without compromising security. We achieve this by issuing to the server random group elements $\hat{t}_1$ from $\mathbb{G}$, and $\hat{t}_0$ from $\mathbb{G}_2$. The server then computes the response to query i as

$$t = e(t_i, \hat{t}_1) \prod_{j \neq i} e(t_j, \hat{t}_0).$$

A remaining technical issue is the fact the in the above discussion we haven't mentioned anything about how the client should remember the new masked value w_i' after an update. Our solution is to compute it pseudo-randomly as $F_k(i, s_i)$ where s_i is a counter that is incremented with each update and is stored together with the payload v_i. This guarantees that a fresh pseudo-random value is used after each update, which in turn allows us to substitute the pseudo-random w_i's by random ones in the security analysis.

6.3 Definition

THE ALGORITHMS. A *verifiable database scheme* $\mathcal{VDB} = (\mathbf{Setup}, \mathbf{Query}, \mathbf{Verify}, \mathbf{Update})$ consists of the four algorithms defined below.

1. $\mathbf{Setup}(DB, n) \rightarrow (PK, SK)$: Based on the security parameter n, the setup algorithm generates a public encoding PK of the database DB, and a secret key SK to be stored at the client.

2. $\mathbf{Query}_{PK}(x) \rightarrow \sigma$: The query processing algorithm takes as input an index x and returns a pair $\sigma = (y, T)$.

3. $\mathbf{Verify}_{SK}(\sigma) \rightarrow y$: Using the client's secret key SK the client verifies the server response σ, and outputs a value y or a special symbol $\perp$.

4. $\mathbf{Update}_{SK}(x, y) \rightarrow t_x'$: Using the client's secret key, the update procedure generates a token t_x' that is sent to the server. The server then uses t_x' to update PK.

SECURITY. Intuitively, a verifiable database scheme is secure if a malicious server cannot persuade the verification algorithm to accept an incorrect output. In other words, when a client submits a query x, the

only response that the server can generate and would be accepted by the verifier is the value v_x that was last written to location x in the database. This can be either the initial value submitted by the client during setup, or the last value that was assigned to index i by invoking the $\mathbf{Update}$ procedure. Below, we formalize this intuition with an experiment, where $poly(\cdot)$ is a polynomial.

Experiment $\mathbf{Exp}_A^{Verif}[\mathcal{VC}, DB, n]$
  $(PK, SK) \leftarrow \mathbf{KeyGen}(DB, n)$;
  For $i = 1, \ldots, \ell = poly(n)$;
    Verify query:
      $(x_i, \sigma_i) \leftarrow A(PK, \beta_1, t_1, \ldots, \beta_{i-1}, t_{i-1})$;
      $\beta_i \leftarrow \mathbf{Verify}_{SK_i}(\sigma_i)$;
    Update query:
      $(x_i, v_{x_i}^{(i)}) \leftarrow A(PK, \beta_1, t_1, \ldots, \beta_{i-1}, t_{i-1})$;
      $(SK_i, t_i) \leftarrow \mathbf{Update}_{SK_{i-1}}(x_i, v_{x_i}^{(i)})$;
  $(x, \sigma) \leftarrow A(PK, \beta_1, t_1, \ldots, x_\ell, \beta_\ell, t_\ell)$;
  $\hat{y} \leftarrow \mathbf{Verify}_{SK}(\sigma)$
  If $\hat{y} \neq \perp$ and $\hat{y} \neq v_x^{(\ell)}$, output '1', else '0';

Above we implicitly assign $SK_i \leftarrow SK_{i-1}$ when a verification query is made, and $v_x^{(i)} \leftarrow v_x^{(i-1)}$ for all x. If an update query is made we implicitly assign $v_x^{(i)} \leftarrow v_x^{(i-1)}$ for all $x \neq x_i$. Essentially, the adversary is given oracle access to verify query responses (and obtain the accepted value if the response was accepted by the verifier), and also oracle access to the update functionality that allows him to obtain tokens that allow him to update the encoding PK of the database. The adversary succeeds if it convinces the verification algorithm to accept on the value for a given index in the database. Our goal is to make the adversary succeed only with negligible probability.

Definition 6.1 (Security)**.** For a verifiable database scheme $\mathcal{VDB}$, we define the advantage of an adversary A in the experiment above as:

$$Adv_A^{Verif}(\mathcal{VC}, D, n) = Prob[\mathbf{Exp}_A^{Verif}[\mathcal{VC}, D, n] = 1] \tag{8}$$

A verifiable database scheme $\mathcal{VDB}$ is secure if for any initial database $DB \in [m] \times \{0,1\}^*$, where $m = poly(n)$, and for any adversary A running in probabilistic polynomial time,

$$Adv_A^{Verif}(\mathcal{VDB}, DB, \lambda) \leq \texttt{negli}(n) \tag{9}$$

where $\texttt{negli}()$ is a negligible function of its input.

6.4 Construction

We construct a verifiable database scheme (VDB) based on any pseudo-random function F, and any bilinear computational group scheme.

Protocol Verifiable-Database

Setup. The setup algorithm takes as input a security parameter n and a set S of index-value pairs $(i, v_i) \in [m] \times \mathbb{Z}_{n-1}$. It then generates groups $\mathbb{G}, \mathbb{G}_T$ of order $N = p_1 p_2$, where p_1, p_2 are primes in the range $[2^{n-1}, 2^n - 1]$, and a pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Let $l = \lceil \log(m) \rceil$. Then choose two PRF keys k_1, k_2 randomly, and choose randomly:

$$g_1, h_1 \in_R \mathbb{G}_1,\ g_2, h_2, u_2 \in_R \mathbb{G}_2,\ a, b \in_R \mathbb{Z}_N$$

For each $x \in \{0, \ldots, 2^l - 1\}$, set $r_x = F_{k_1}(x)$, $w_x = F_{k_2}(x, 1)$, and $s_x = 1$. Then set $c_x = v_x$ if $(x, v_x) \in S$, and $c_x = 0$ otherwise. Then, set

$$t_i \leftarrow g_1^{r_i + ac_i + bs_i} g_2^{w_i} \quad \forall\, 0 \le i \le 2^l - 1$$
$$w = \sum_{i=0}^{2^l - 1} w_i,\ T_w \leftarrow e(g_2, u_2)^w$$

Set $\hat{t}_0 = u_2$ and $\hat{t}_1 = h_1 h_2$. The public key is $PK = \big((\hat{t}_0, \hat{t}_1, s_0, t_0, \ldots, s_{2^l-1}, t_{2^l-1}), S\big)$. The private key is $SK = (a, T_w, k_1, k_2)$.

Query. The query processing algorithm takes as input the public key *PK* and a query $x \in \{0,1\}^l$. Then compute

$$T \leftarrow e(t_x, \hat{t}_1) \cdot e\left(\prod_{i=0, i \neq x}^{2^l - 1} t_i, \hat{t}_0\right)$$

If $(x, v_x) \in S$ set $y \leftarrow v_x$, and set $y \leftarrow 0$ otherwise. Output (T, y, s_x).

Verification. The verification algorithm takes as input a private key *SK*, a query x, and server output (T, y, s_x). The verifier then computes $r_x = F_{k_1}(x)$, $w_x = F_{k_2}(x, s_x)$ and checks

$$T \stackrel{?}{=} T_w \cdot e(g_1^{r_x + a \cdot y + b \cdot s_x} g_2, h_1 u_2^{-w_x} h_2^{w_x})$$

If the equality holds, the verifier outputs b. Otherwise, the verifier outputs $\perp$.

Update. The update algorithm takes as input a pair $(x, \delta) \in [m] \times \mathbb{Z}_{n-1}$. It then submits x as a query to the server, and verifies the response (T, y, s_x). Then, set $w'_x = F_{k_2}(x, s_x + 1) - F_{k_2}(x, s_x)$, and sets

$$T_w \leftarrow T_w \cdot e(g_2^{w'_x}, u_2) \quad \text{and} \quad t'_x \leftarrow g_1^{a \cdot \delta + b} g_2^{w'_x}$$

The server is then given t'_x, and updates the public key by setting $t_x \leftarrow t_x \cdot t'_x$ and $s_x \leftarrow s_x + 1$.

CORRECTNESS. Correctness follows from properties of bilinear maps over composite order groups:

$$\begin{aligned} T = e(t_x, \hat{t}_1) e(\prod_{i \neq x} t_i, \hat{t}_0) &= e(g_1^{r_x + ac_x + bs_x}, g_2^{w_i}, h_1 h_2) \prod_{i \neq x} e(g_1^{r_i + ac_i + bs_i}, g_2^{w_i}, u_2) \\ &= e(g_1^{r_x + ac_x + bs_x} g_2, h_1 u_2^{-w_x} h_2^{w_x}) \prod_{i} e(g_1^{r_i + ac_i + bs_i}, g_2^{w_i}, u_2) \\ &= e(g_1^{r_x + ac_x + bs_x} g_2, h_1 u_2^{-w_x} h_2^{w_x}) \cdot T_w \end{aligned}$$

6.5 Analysis

Theorem 6.2. *Protocol* Verifiable-Database *is secure according to the Definition 6.1*

Our proof proceeds via a sequence of games:

Game 0. Game 0 is the original security experiment, played with the adversary A.

Game 1. Game 1 proceeds as Game 0, except that the private key and the masking values w_x are generated differently, and the verification procedure is modified. Specifically, the values $r_0, \ldots, r_{2^l-1}$ are chosen randomly from $\mathbb{Z}_N$ and the private key is set to

$$SK \leftarrow (a, T_w, k, r_0, \ldots, r_{2^l-1})$$

The verification is then performed directly using the values $\{r_i\}_{0 \le i \le 2^l-1}$ (instead of first computing r_x as $r_x = F_{k_1}(x)$).

The masking values w_x are generated randomly, rather than as outputs of the PRF. Specifically, let l be an upper bound on the number of update queries made by the adversary. The challenger in Game 1 generates, for each $x \in [m]$, l random values $w_x^{(1)}, \ldots, w_x^{(l)}$. During setup and updates, the value w_x^i is used instead of $F_{k_2}(x, i)$.

Game 2. Game 2 proceeds as Game 1, except that the following values are generated differently. In the setup phase, the challenger generates:

$$g_{12}, h_{12} \in_R \mathbb{G}_1, \ \hat{g}_2, u_2, \hat{h}_2 \in_R \mathbb{G}_2$$

Then the challenger sets

$$t_i \leftarrow g_{12}^{r_i + ac_i + bs_i} \hat{g}_2^{w_i}, \quad \hat{t}_0 \leftarrow u_2, \quad \hat{t}_1 \leftarrow h_{12}\hat{h}_2$$
$$T_{w,x} \leftarrow e(t_x, \hat{h}_2) \prod_{i \ne x} e(t_i, u_2)$$

Verification is modified to perform the following check:

$$T \stackrel{?}{=} T_{w,x} \cdot e(g_{12}^{r_x + ay + bs_x} \hat{g}_2^{w_x}, h_{12})$$

Updates are now performed by setting:

$$T_{w,x} \leftarrow T_{w,x} \cdot e(\hat{g}_2^{w'_x}, \hat{h}_2), \quad T_{w,y} \leftarrow T_{w,y} \cdot e(\hat{g}_2^{w'_x}, u_2) \quad \text{for} \quad y \ne x$$
$$t'_x \leftarrow g_{12}^{a\delta + b} \hat{g}_2^{w'_x}$$

Game 3. Game 3 proceeds identically to Game 2, except that the group elements g_{12}, h_{12} are chosen from the group $\mathbb{G}$ instead of $\mathbb{G}_1$.

Game 4. Game 4 proceeds identically to Game 3, except that the group element $\hat{g}_2$ is chosen from the group $\mathbb{G}$ instead of $\mathbb{G}_2$.

The following claim follows straightforwardly from the security of the pseudorandom function.

Claim 6.3. *No adversary can distinguish between Games 0 and 1 with advantage more than* $\varepsilon_{\mathsf{PRF}}$.

Claim 6.4. *Games 1 and 2 proceed identically.*

Proof. Let us denote $\delta = r_x + ay + bs_y$ and $\gamma_i = r_i + ac_i + bs_i$. The check performed by the challenger can be written as follows:

$$\begin{aligned} T &\stackrel{?}{=} e(g_{12}^{\delta} \hat{g}_2^{w_x}, h_{12}) \cdot e(g_{12}^{\gamma_x} \hat{g}_2^{w_x}, \hat{h}_2) \prod_{i \ne x} e(g_{12}^{\gamma_i} \hat{g}_2^{w_i}, u_2) \\ &= e(g_{12}, h_{12})^{\delta} \cdot e(\hat{g}_2, \hat{h}_2)^{w_x} \prod_{i \ne x} e(\hat{g}_2, u_2)^{w_i} \\ &= e(g_{12}, h_{12})^{\delta} \cdot e(\hat{g}_2, \hat{h}_2)^{w_x} e(\hat{g}_2, u_2)^{\sum_{i \ne x} w_i} \end{aligned}$$

On the other hand, the check performed by the challenger in Game 1 is:

$$\begin{aligned} T &\stackrel{?}{=} e(g_2, u_2)^{\sum w_i} \cdot e(g_1^{\delta} g_2, h_1 u_2^{-w_x} h_2^{w_x}) \\ &= e(g_2, u_2)^{\sum_{i \neq x} w_i} \cdot e(g_1, h_1)^{\delta} e(g_2, h_2)^{w_x} \end{aligned}$$

Finally, we observe that the tuples $(g_{12}, h_{12}, \hat{g}_2, \hat{h}_2, u_2)$ and $(g_1, h_1, g_2, h_2, u_2)$ are identically distributed. □

Claim 6.5. *Games 2 and 3 are indistinguishable if the Bilinear Subgroup Projection assumption holds.*

Proof. We use the adversary A to construct a distinguisher B that breaks Assumption 1 (by distinguishing a random member of $\mathbb{G}_2$ from a random member of $\mathbb{G}$). B is given a challenge h_c. B then generates randomly $g_{12}, h'_{12} \in_R \mathbb{G}$, sets $h_{12} = h'_{12} h_c$ and generates all other values as in Game 3.

We shall first argue that the public key is distributed identically in games 2 and 3. In game 3, we can write $g_{12} = g_1 g_2$, $h_{12} = h_1 h_2$, $\hat{g}_2 = g_2^{\beta}$, where $g_1, h_1 \in \mathbb{G}_1$ and $g_2, h_2 \in \mathbb{G}_2$, and $\beta \in \mathbb{Z}_N$. We then get:

$$\gamma_i \leftarrow r_i + a c_i + b s_i, \quad t_i = g_{12}^{\gamma_i} \hat{g}_2^{w_i} = g_1^{\gamma_i} g_2^{\gamma_i + \beta w_i}, \quad \hat{t}_0 = u_2, \quad \hat{t}_1 = h_1 (h_2 \hat{h}_2)$$

Since w_i are uniformly random, so are the values $\gamma_i + w_i$. Similarly, to update t_x, for some x, in game 3 the adversary is given $t'_x = g_{12}^{a\delta + b} \hat{g}_2^{w'_x} = g_1^{a\delta + b} g_2^{a\delta + b + \beta w'_x}$. The exponent of g_1 is the same as in game 2, while the exponent of g_2 is uniformly random (since w'_x is random), and therefore has the same distribution as the exponent of g_2 in game 2.

Consequently, the view of the adversary is identical in games 2 and 3, unless he submits a query x_i, and a certificate $\sigma_{y',i}$, such that $\sigma_{y',i}$ is accepted in game 2 but rejected in game 3, or vice versa. We now consider each of these two cases:
Case 1: let us denote by B_i the event that $\sigma_{y',i} = (y', T)$ is accepted in Game 2, but rejected in Game 3, and it is the ***first*** such query. The values computed by the challenger, and compared to T, in games 2 and 3 respectively are:

$$T_2 = e(g_1^{r_x + a y' + b s_{y'}}, h_1) e(g_2^{\gamma_x + \beta w_x}, h_2) e(g_2^{\gamma_x + \beta w_x}, \hat{h}_2) \prod_{i \neq x} e(t_i, u_2)$$

$$T_3 = e(g_1^{r_x + a y' + b s_{y'}}, h_1) e(g_2^{r_x + a y' + b s_{y'} + \beta w_x}, h_2) e(g_2^{\gamma_x + \beta w_x}, \hat{h}_2) \prod_{i \neq x} e(t_i, u_2)$$

Event B_i implies that $T = T_2$. Therefore, we can compute:

$$W_0 = (T_3 / T)^{(a(y' - c_x) + b(s_{y'} - s_x))^{-1}} = e(g_2, h_2) \tag{10}$$

We assume that $(a(y' - c_x) + b(s_{y'} - s_x))^{-1} \mod N$ exists. Otherwise, we can easily find the factors p_1, p_2 of N. Suppose that event B_i occurs with probability ε. Therefore, the adversary A can be used to break the BSGP assumption with advantage at least ε / n.

Case 2: let us denote by B'_i the event that $\sigma_{y',i} = (y', T)$ is accepted in Game 3, but rejected in Game 2, and it is the ***first*** such query. By applying Claim 6.6 and Claim 6.7 we obtain that this event only occurs with negligible probability. □

Claim 6.6. *The difference between the probabilities that the challenger accepts an invalid certificate in Games 3 and 4 is negligible.*

Proof. The proof proceeds by a simple reduction to Assumption 2. Note that in games 3 and 4 all group elements are either chosen randomly from $\mathbb{G}_2$ or $\mathbb{G}$, and that the reduction has access to generators of both these groups. The reduction receives a challenge g_c and assigns $\hat{g}_2 \leftarrow g_c$. Now, if g_c is a random member of $\mathbb{G}_2$, we obtain a perfect simulation of Game 3, while if it is a random member of $\mathbb{G}$ we get a simulation of Game 4.

Let us denote by $B_i^{(3)}$ and $B_i^{(4)}$ the events that the ith query $\sigma_{y',i}$ is invalid (i.e. $y' \neq y$), and is accepted by the challenger in Game 3 and 4 respectively. Consider any $1 \leq i \leq l$ for which $\Pr[B_i^{(3)}] - \Pr[B_i^{(4)}] = \varepsilon$. We can construct a distinguisher D that simulates A up to the ith query. Then, if $y' \neq y$ but the query is accepted, D outputs 1. Otherwise, D tosses a fair coin and outputs the outcome. We then get

$$\begin{aligned}\Pr[D(g_c) = 1 | g_c \in_{\mathrm{R}} \mathbb{G}_2] - \Pr[D(g_c) = 1 | g_c \in_{\mathrm{R}} \mathbb{G}] &= \Pr[D(g_c) = 1 | g_c \in_{\mathrm{R}} \mathbb{G}_2,\ B_i^{(2)}] \Pr[B_i^{(3)}] \\ &\quad - \Pr[D(g_c) = 1 | g_c \in_{\mathrm{R}} \mathbb{G},\ B_i^{(3)}] \Pr[B_i^{(4)}] \\ &= \Pr[B_i^{(3)}] - \Pr[B_i^{(4)}]\end{aligned}$$

□

Claim 6.7. *Let $B_i^{(4)}$ be the event that in Game 4 A submits a $\sigma_{y',i}$ where $y' \neq v_i$, and the verifier accepts, and it is the first such query. Then, $\Pr[B_i^{(4)}] \leq l/p'$ where $p' = \min\{p_1, p_2\}$ and $N = p_1 p_2$ is the size of the group $\mathbb{G}$.*

Proof. We show that in order for event $B_i^{(4)}$ to occur the adversary A must win in the experiment described in Lemma 5.4. The claim then follows immediately from Lemma 5.4. Let us recall the check that is performed by the challenger in Game 4 to verify a certificate $\sigma_{y',i} = (y', T, s_x)$: let $\hat{g}_2 = g_1^{\alpha} g_2^{\beta}$

$$T \stackrel{?}{=} T_{w,x} \cdot e(g_{12}^{r_x + ay + bs_x} \hat{g}_2^{w_x^{(i)}}, h_{12})$$

We now describe an inefficient reduction to the game of Lemma 5.4 (recall that the statement of Lemma 5.4 holds even against computationally unbounded adversaries). The reduction simulates the adversary A as follows. A outputs an initial DB $(v_1, \ldots, v_m)$. The reduction then queries the MACing oracle on inputs $(v_1, 1), \ldots, (v_m, 1)$ to obtain MACs $(\tau_1, \ldots, \tau_m)$. The reduction then generates group elements

$$\begin{aligned}&g_{12}, h_{12}, \hat{g}_2, u_2, \hat{h}_2 \\ &\text{and sets } t_i \leftarrow (g_{12}\hat{g}_2)^{\tau_i}\end{aligned}$$

The reduction then simulates A until A submits the ith query $(x_i, \sigma_{y,i})$. The simulation is performed as follows: when A submits a query $(x_j, \sigma_{y,j})$ for $j < i$, the reduction performs two checks: (i) if $y \neq v_{x_j}$ then the reduction responds with $\perp$. (ii) if $y = v_{x_j}$ then the reduction checks

$$T \stackrel{?}{=} T_{w,x} e((g_{12}\hat{g}_2)^{\tau_j}, h_{12})$$

and responds to the query with y if the check passes and with $\perp$ otherwise.

To handle an update query (i, v_i') the reduction queries the MAC oracle on $(v_i', s_i + 1)$ to obtain a MAC τ_i'. Note that $s_i + 1$ never appeared before as a counter value (otherwise the adversary successfully forges a MAC during the query part of the update procedure), therefore, in Game 4, a fresh value w_i' would have been generated, and so obtaining a fresh MAC yields the correct distribution. It then sets $s_i \leftarrow s_i + 1$, $T_{w,x} \leftarrow T_{w,x} t_i^{-1} (g_{12}\hat{g}_2)^{\tau_i}$. The reduction then sends $t_i' = (g_{12}\hat{g}_2)^{\tau_i' - \tau_i}$ to A as a response. Finally, when A submits the ith query $(x_i, \sigma_{y,i})$ the reduction parses $\sigma_{y,i} = (y, T)$, computes $T' = T/T_{w,x}$ and extracts the discrete logarithm τ of T' relative to $e(g_{12}\hat{g}_2, h_{12})$. Suppose that event $B_i^{(4)}$ occurs, then τ is a valid MAC under the same key as τ_i. □

7 Proofs of Retrievability

Assume a client stores a (potentiall long) file on a server, but wants some assurance that the file is indeed retrievable when needed. The client could periodically request the file from the server and check it against a short fingerprint that the client stored, but this would require large communication (to send the file) and computation (to recompute the fingerprint). The goal of an efficient proof of retrievability is to have short communication protocol, and an efficient computation (at least for the client) which will assure the client that the server is still storing the correct file. Formally this is defined by enforcing an *extraction* property: if at the end of the protocol the server convinces the client with non-negligible probability, then there exists an *extractor* machine that interacting with the server is able to efficiently reconstruct the file. We refer to [58] for a formal definition.

A proof of retrievability protocol usually goes like this: the client issues a *query* to the server, which computes an answer based on the query and the file, and finally the client performs some verification procedure on the answer. Usually the important parameters are: the communication required by the query and the answer, the computation time required by the client and the server, and the storage overhead of the server. Denoting by n the security parameter, let f be the number of n-bit blocks required to write the file. A proposal by Shacham and Waters [58] has queries of size $O(n^2)$, computation time by the client and the server $O(n)$ (if we count basic operations such as modular multiplication as a unit of time), and an overhead of 4 in terms of storage (the server must store $4f$ blocks, since the file is encoded using an erasure code that doubles its size, and for each block the server must also store its MAC). The protocol in [22] improve on the communication ($O(n)$) and storage parameters (the server stores only the file, and a MAC for each block) but then requires the server and the client to compute over the entire file ($O(f)$ computation).

We present a simple solution based on our verifiable delegation of polynomials, that has optimal communication (query and answer are $O(n)$ size) and client computation ($O(n)$ or $O(n \log f)$ depending on the underlying security assumption) and requires no storage overhead. In our solution only the server is required to compute over the entire file.

Our protocol Retrievability-Proof works as follows. The client encodes the file as a polynomial $F(x)$ of degree d (each block representing a coefficient in $\mathbb{Z}_p$ where p is the order of the group used in our verifiable delegation scheme, determined by the security parameter). It then delegates the computation of $F(x)$ to the server (according to protocol Delegate-Polynomial described in Section 5). The proof of retrievability consists of the client and the server engaging in our verifiable delegation protocol Delegate-Polynomial over a random point r: the client sends r and the server returns the value $F(r)$ together with a proof of its correctness. The client accepts if it accepts the proof that $F(r)$ is the correct value.

Theorem 7.1. *Protocol* Retrievability-Proof *is a secure proof of retrievability.*

The security follows almost immediately from the security of our delegation scheme. Indeed if the client accepts at the end of our protocol then it means that the server returned the correct $F(r)$ value. Therefore if the server convinces the client with non-negligible probability δ then in time $O(d/\delta)$ we can obtain $d+1$ correct points on the polynomial and extract the file by polynomial interpolation.

Acknowledgments

Rosario Gennaro's research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the US Government, the UK Ministry of

Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints of this work for Government purposes, notwithstanding any copyright notation hereon.

References

[1] M. Ajtai, "The invasiveness of off-line memory checking", in *STOC*, pp. 504–513, 2002.

[2] Amazon Elastic Compute Cloud. Online at http://aws.amazon.com/ec2.

[3] B. Applebaum, Y. Ishai, E. Kushilevitz, "From Secrecy to Soundness: Efficient Verification via Secure Computation", in *ICALP*, pp.152–163, 2010.

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores", in *ACM CCS*, pp. 598–609, 2007.

[5] L. Babai, "Trading group theory for randomness", in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, (New York, NY, USA), pp. 421–429, ACM, 1985.

[6] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya, "Incentivizing outsourced computation", in *Proceedings of the Workshop on Economics of Networked Systems (NetEcon)*, (New York, NY, USA), pp. 85–90, ACM, 2008.

[7] S. Benabbas, R. Gennaro, Y. Vahlis, "Verifiable Delegation of Computation over Large Datasets", *Cryptology ePrint Archive, Report 2011/132*, http://eprint.iacr.org/.

[8] M. Bellare, B. Waters, and S. Yilek, "Identity-based encryption secure against selective opening attack." To appear in TCC 2011, 2011.

[9] M. Ben-Or, S. Goldwasser and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation", in *STOC*, pp.1–10, 1988.

[10] M. Blum, W. Evans, P. Gemmell, S. Kannan, M. Naor, "Checking the correctness of memories", Foundations of Computer Science, Annual IEEE Symposium on, pp. 90-99, 32nd Annual Symposium of Foundations of Computer Science (FOCS 1991), 1991

[11] D. Boneh and X. Boyen, "Efficient selective-id secure identity-based encryption without random oracles", in *EUROCRYPT*, pp. 223–238, 2004.

[12] D. Boneh and X. Boyen, "Short signatures without random oracles and the sdh assumption in bilinear groups", *J. Cryptology*, vol. 21, no. 2, pp. 149–177, 2008.

[13] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts", in *TCC*, pp. 325–341, 2005.

[14] D. Boneh, H. Montogomery, and A. Raghunathan, "Algebraic pseudorandom functions with improved efficiency from the augmented cascade", in *Proc. of ACM CCS'10*, 2010.

[15] D. Boneh, A. Sahai, and B. Waters, "Fully collusion resistant traitor tracing with short ciphertexts and private keys", in *EUROCRYPT*, pp. 573–592, 2006.

[16] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials", in *CRYPTO*, pp. 6176, 2002.

[17] D. Chaum, C. Crepeau and I. Damgard, "Multiparty unconditionally secure protocols", in *STOC*, pp.11–19, 1988.

[18] D. Chaum and T. Pedersen, "Wallet databases with observers", in *Proceedings of CRYPTO*, 1992

[19] J. H. Cheon, "Security analysis of the strong diffie-hellman problem", in *EUROCRYPT*, pp. 1–11, 2006.

[20] K.M. Chung, Y. Kalai, S.P. Vadhan, "Improved Delegation of Computation Using Fully Homomorphic Encryption", in *CRYPTO*, pp. 483–501, 2010.

[21] Y. Desmedt, "Threshold Cryptography", in *Encyclopedia of Cryptography and Security*, 2005.

[22] Y. Dodis, S.P. Vadhan, D. Wichs, "Proofs of Retrievability via Hardness Amplification", in *TCC*, pp.109-127, 2009.

[23] Y. Dodis and A. Yampolskiy, "A verifiable random function with short proofs and keys", in *Public Key Cryptography*, pp. 416–431, 2005.

[24] C. Dwork, M. Naor, G. N. Rothblum, V. Vaikuntanathan, "How Efficient Can Memory Checking Be?", in *TCC*, pp. 503–520, 2009.

[25] D. M. Freeman, "Converting pairing-based cryptosystems from composite-order groups to prime-order groups", in *EUROCRYPT*, pp. 44–61, 2010.

[26] P. Gemmell, M. Naor, "Codes for Interactive Authentication", in *CRYPTO*, pp. 355–367, 2003.

[27] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers", in *CRYPTO*,pp. 465–482, 2010.

[28] C. Gentry, "Fully homomorphic encryption using ideal lattices", in *Proceedings of the ACM Symposium on the Theory of Computing (STOC)*, 2009.

[29] O. Goldreich, S. Micali and A. Wigderson, "How to play ANY mental game", in *STOC*, pp.218–229, 1987.

[30] 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.

[31] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems", *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[32] P. Golle and I. Mironov, "Uncheatable distributed computations", in *Proceedings of the RSA Conference*, 2001.

[33] E. Hall, C. Jutla, "Parallelizable Authentication Trees", in *Selected Areas in Cryptography*, pp.95-109, 2005.

[34] C. Hazay and Y. Lindell, "Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries", in *J. Cryptology*, 23(3): 422-456 (2010)

[35] S. Hohenberger and A. Lysyanskaya, "How to securely outsource cryptographic computations", in *Proceedings of TCC*, 2005.

[36] A. Juels, B.S. Kaliski Jr., "Pors: proofs of retrievability for large files", in*ACM Conference on Computer and Communications Security*, pp.584-597, 2007

[37] J. Kilian, "A note on efficient zero-knowledge proofs and arguments (extended abstract)", in *Proceedings of the ACM Symposium on Theory of computing (STOC)*, (New York, NY, USA), pp. 723–732, ACM, 1992.

[38] J. Kilian, "Improved efficient arguments (preliminary version)", in *Proceedings of the International Cryptology Conference on Advances in Cryptology*, (London, UK), pp. 311–324, Springer-Verlag, 1995.

[39] A. Lewko, M. Lewko, and B. Waters, "How to leak on key updates." STOC, 2011.

[40] A. Lewko, Y. Rouselakis, and B. Waters, "Achieving leakage resilience through dual system encryption." TCC, 2011.

[41] A. B. Lewko and B. Waters, "Efficient pseudorandom functions from the decisional linear assumption and weaker variants", in *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, (New York, NY, USA), pp. 112–120, ACM, 2009.

[42] A. Lewko and B. Waters, "Unbounded hibe and attribute-based encryption." Eurocrypt, 2011.

[43] C. Martel and G. Nuckolls and P. Devanbu and M. Gertz and A. Kwong and S. G. Stubblebine, "A general model for authenticated data structures", *Algorithmica,* vol. 39, no. 1, pp. 21–31, 2004.

[44] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function", in *CRYPTO*, pp. 369–378, 1987.

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

[46] S. Micali, M.O. Rabin, J. Kilian, "Zero-Knowledge Sets", in *FOCS*, pp.80–91, 2003.

[47] S. Mitsunari, R. Sakai, and M. Kasahara, "A new traitor tracing", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 85, no. 2, pp. 481–484, 2002.

[48] F. Monrose, P. Wyckoff, and A. Rubin, "Distributed execution with remote audit", in *Proceedings of ISOC Network and Distributed System Security Symposium (NDSS)*, Feb. 1999.

[49] L. Nguyen, "Accumulators from bilinear pairings and applications", in *CT-RSA 2005*, vol. 3376, pp. 275292, 2005

[50] M. Naor and K. Nissim, "Certificate revocation and certificate update", *USENIX Security*, pp. 17–17, 1998.

[51] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions", *J. ACM*, vol. 51, pp. 231–262, March 2004.

[52] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions", in *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, (Washington, DC, USA), pp. 458–, IEEE Computer Society, 1997.

[53] M. Naor and G. N. Rothblum, "The Complexity of Online Memory Checking", in *FOCS*, pp.573–584, 2005.

[54] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes", in *EUROCRYPT*, pp. 223–238, 1999.

[55] C. Papamanthou, R. Tamassia, "Time and space efficient algorithms for two-party authenticated data structures", in *ICICS 07*, vol. 4861, pp. 115, 2007

[56] C. Papamanthou, R. Tamassia, N. Triandopoulos, "Authenticated hash tables", in CCS, pp. 437448, Oct 2008

[57] C. Papamanthou and R. Tamassia and N. Triandopoulos, "Optimal Authentication of Operations on Dynamic Sets", *Cryptology ePrint Archive, Report 2010/455*, http://eprint.iacr.org/.

[58] H.Shacham and B. Waters, *Compact Proofs of Retrievability*, in *ASIACRYPT*, pp.90–107, 2008.

[59] S. Smith and S. Weingart, "Building a high-performance, programmable secure coprocessor", *Computer Networks (Special Issue on Computer Network Security)*, vol. 31, pp. 831–960, 1999.

[60] Sun Utility Computing. Online at http://www.sun.com/service/sungrid/index.jsp.

[61] R. Tamassia, "Authenticated data structures, *European Symp. on Algorithms,* pp. 2–5, 2003.

[62] R. Tamassia, N. Triandopoulos, "Certification and authentication of data structures", in Proc. Alberto Mendelzon Workshop on Foundations of Data Management. Cite-seer (2010)

[63] A. Yao, "Protocols for secure computations", in *FOCS*, pp. 1982.

[64] B. S. Yee, *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.

[65] B. Waters, "Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions", in *CRYPTO*, pp. 619–636, 2009.

Protocol	π'	π_0	π_1	π_2	π_3	$\cdots$
Content	$\{15, 5, 13, 2, 145\}$	$\{12\}$	$\{1, 3\}$	$\emptyset$	$67, 46, 6, 235, 23, 64, 76, 0$	$\cdots$

Figure 1: A possible state of the protocol for implementing insertions (§A.1.) The database was initialized containing the keywords $15, 5, 13, 2, 145$ with the rest added later using "Insert" operations.

A Generic Updates

In this section we present a generic way to add insertion and deletions to any (static) implementation of verifiable keyword matching.

A.1 Generic Insertion

In this section we present a generic way to add insertion to any implementation of verifiable keyword matching. We assume we have a way to implement a *static* keyword matching with setup time $O(f(n))$ and query time $O(g(n))$ for the client and we show a protocol with setup time $O(f(n))$, query time $O(g(n) \log n)$ and insertion time $O(f(n) \log n/n + g(n) \log n)$. There two down sides however, firstly the insertion time is *amortized running time* and secondly the insertion takes 2 interactions between the client and the server.[8] Here n is the total number of elements in the database. The server's runtime is only increased by a factor of $\log n$ for queries, and a factor of n (resp. $\log n$) for insertions in worst case (reps. amortized.)

The outline of the scheme is as follows. We will call the original (static) protocol Π. At any point in time the client and server will have multiple copies of this protocol instantiated with the set of keywords currently present in the database distributed among the protocols. We will call these multiple copies $\pi', \pi_0, \pi_1, \ldots, \pi_{\log n}$. The following invariants will remain true through out any interaction between the server and the client. Firstly, the set of keywords the database was initialized with are always stored in π'. Secondly, any other keywords that the client adds to the database is stored in exactly one of $\pi_0, \pi_1, \ldots$. Thirdly, for any i and at any point in time either π_i does not contain any keywords or it contains 2^i keywords; we call the former (resp. later) an active (resp. inactive) protocol. Figure 1 shows a possible state of the database. The client will keep track of the private information associated with each of the Π protocols and the total number of "Insert" operations so far (called m.)

Notice that as long as the above invariants hold m will completely determine which of the π_i databases are empty and which are nonempty. In fact the empty databases correspond to the bits in the binary representation of m that are 0. Notice that initializing the database will take runtime $O(f(n) + f(1) \log n) = O(f(n))$ for the client. To search the client and server run $\log n+1$ parallel search protocols on $\pi_0, \ldots, \pi_{\log n}$ and π'. Given that each of these databases has size at most $O(n)$ the runtime of the server and the client is multiplied by $\log n$. The Insert operation is a bit more complicated.

We will describe the three operations below. Below, we assume that the $\mathbf{QueryGen}$ step of the static protocol Π just outputs x like the ones we developed in previous sections. This assumption is not strictly required but it clarifies the presentation and simplifies notation. What is required is for the server to prove all the content of S_i to the client. This is always attainable in one extra round of communication with the server first sending S_i to the client and the client asking for the status of all the elements of S_i in parallel from the server. But when the output of the $\mathbf{QueryGen}$ method can be easily generated for the members of S_i by the server (like assumed in the following presentation of the protocol) this extra pass is not required.

[8]The first interaction starts with the client just sending an empty message and the server replying and the second has data in both directions.

$\mathbf{Setup}(S)$: Let $n = |S|$. Run $(PK_0, SK_0) = \mathbf{Setup}(\emptyset)$, ..., $(PK_{\log n}, SK_{\log n}) = \mathbf{Setup}(\emptyset)$ to generate $\log n + 1$ empty databases corresponding to $\pi_0, \ldots, \pi_{\log n}$ and $(PK', SK') = \mathbf{Setup}(S)$ to generate a database corresponding to π'. Output $PK = ((PK', S), (PK_0, \emptyset), \ldots, (PK_{\log n}, \emptyset))$ and $SK = (SK', SK_0, \ldots, SK_{\log n})$.

$\mathbf{QueryGen}(SK, x)$: Output $(\sigma_x, \tau_x) = (x, x)$.

$\mathbf{CompQuery}(PK, \sigma_x)$: Parse PK as $((PK', S), (PK_0, \emptyset), \ldots, (PK_{\log n}, \emptyset))$. Compute $z' = \mathbf{CompQuery}(PK', \sigma_x)$ and $z_i = \mathbf{CompQuery}(PK_i, \sigma_x)$ for all $0 \leq i \leq \log n$. Output $z = (z', z_0, \ldots, z_{\log n})$.

$\mathbf{Verify}(SK, \tau_x, z)$: Parse SK as $(SK', SK_0, \ldots, SK_{\log n})$, and z as $(z', z_0, \ldots, z_{\log n})$. Compute $y' = \mathbf{Verify}(SK', \tau_x)$ and $y_i = \mathbf{Verify}(SK_i, \tau_x)$, for all $0 \leq i \leq \log n$. Output y defined as,

$$y = \begin{cases} \perp & y' = \perp \text{ or } y_i = \perp \text{ for some } i, \\ True & \text{otherwise if } y' = True \text{ or } y_i = True \text{ for some } i, \\ False & \text{otherwise.} \end{cases}$$

That is, if any of the verifications failed, output $\perp$, otherwise output $x \in S$ if x was found in any of the databases $S', S_0, \ldots, S_{\log n}$.

$\mathbf{InsertGen}(SK, x)$: Output $(\sigma_x, \tau_x) = (x, x)$.

$\mathbf{CompInsert1}(PK, \sigma_x)$: Let l be the least significant zero bit of m in base two. Send the content of $S_0, \ldots, S_{l-1}$ to the client. Also for each element run $\mathbf{CompQuery}$ for that element and

$\mathbf{InsertGet}1(SK, \tau_x, z)$: Let l be the least significant zero bit of m in base two. Parse z as sets $S_0, \ldots, S_{l-1}$ and $2^l - 1$ outputs of $\mathbf{CompQuery}$. Run $\mathbf{Verify}$ on all the $2^l - 1$ outputs (together with the corresponding element in the received S_i) and output $\perp$ if any of the runs resulted in $False$ or $\perp$. Let $(PK_i, SK_i) = Setup(\emptyset)$ for $0 \leq i < l$. Let $(PK_l, SK_l) = Setup(S_0 \cup \cdots \cup S_{l-1} \cup \{x\})$. Send $(PK_0, \ldots, PK_l)$ to the server. Update the first $l + 1$ elements of SK to $SK_0, \ldots, SK_l$.

$\mathbf{CompInsert2}(PK, \sigma_x, z')$: Parse z' as $(PK_0, \ldots, PK_l)$ and update PK accordingly.

It is not hard to see that the above operations maintain the invariants. To analyze the amortized runtime of the Insert operation notice that in a sequence of M inserts $\mathbf{Setup}$ is ran exactly $\lceil M/2^{i+1} \rceil$ times on π_i. Furthermore, before each such $\mathbf{Setup}$ there are exactly 2^i $\mathbf{Verify}$ operations. So the overall client runtime will be,

$$\leq \sum_{i=0}^{\log M} (f(2^i) + 2^i g(2^i)) \lceil M/2^{i+1} \rceil \leq \sum_{i=0}^{\log M} (f(2^i) + 2^i g(2^i)) M/2^i = 2 \log M f(M) + M \log M g(M)),$$

where we have assumed that $f(\cdot)$ is grows at least linearly. In other words if n is the total number of elements in the database,

$$\text{Amortized insert time} = O(\frac{f(n) \log n}{n} + g(n) \log n).$$

The exact same analysis shows that the amount of communication between the server and the client behaves in the same way.

A.2 Generic Deletion

In this section we present a generic way to add deletions to any implementation of verifiable keyword matching that already supports Insertion. We assume we have a way to implement a keyword matching with setup time $O(f(n))$, query time $O(g(n))$ and Insertion time $O(h(n))$ for the client and we show a protocol with setup time $O(f(n))$, query time $O(g(n)\log n)$, and *amortized* insertion and deletion time $O(g(n)\log n + h(n) + f(n)\log n/n)$.[9] Here n is the total number of elements in the database. The server's runtime is only increased by a factor of $\log n$ for queries, and a factor of n (resp. $\log n$) for insertions and deletions in worst case (reps. amortized.) Depending on the implementation we are trying to add deletions to insertions and deletions might take one more round in the resulting implementation than insertions in the original implementation. However if we are using the output of the previous subsection the number of rounds of communication does not increase. Thus, applying the construction of this subsection after that of subsection A.1 we have a way to add insertion and deletion to any verifiable keyword matching protocol (assuming the same condition as in the previous section) which increases the number of rounds of communication by just one.

The outline of the scheme is as follows. We will call the original (static) protocol Π. At any point in time the client and server will have multiple copies of this protocol instantiated which we call $\pi_1, \ldots, \pi_k$. We call the set of keywords stored via these protocols $S_1, \ldots, S_k$ and the set of keywords in the database S. The idea is to store the keywords which are inserted into the database exactly once in S_1, the keywords which are deleted in S_2, the ones which are added again after getting deleted in S_3 and so on. In other words for all i, $S_i \supseteq S_{i+1}$ and $S = S_1 \backslash S_2 \cup S_3 \backslash \cdots$. There are two problems with this idea however, first of all the combined size of S_i's, and hence the running times, will be a function of the total update operations done on the database and *not* its size. But perhaps more importantly, the number of the sets S_i's (and π_i's) we need will be the number of times an element can get deleted and reinserted which is unbounded. To solve these two issues we also add the condition that at any point in time for all i, $|S_i| \geq 2|S_{i+1}|$. If any S_i gets bigger than half the size of the one preceding it then the server simply sends the of contents S_j for all $j >= i-1$ to the client and the client reinitializes S_{i-1} to $S_{i-1} \backslash S_i \cup S_{i+1} \backslash \cdots$. It is not hard to see that the total sizes of S_j for all $j >= i-1$ is at most $4|S_j|$ while this total size will be reduced to at most $3|S_j|/4$ after the update. The amortized runtime will then follow in a straight forward manner using the potential function method by defining a potential function $\Phi(S_1, S_2, \ldots) = \sum_{i>1} |S_i| + f(|S_i|)$. We leave the details of the protocol as well as the proof of its security for the full version of the paper.

[9] as a technical condition we also need (the reasonable condition) that $f(n)$ is at most polynomial in n.