Applications of Polynomial Properties to Verifiable Delegation of Computation and Electronic Voting

Sandra Guasch Paz Morillo Marc Obrador

December 30, 2012

Abstract

This paper presents some proposals of protocols for two types of schemes such as verifiable delegation of computation and remote electronic voting, based on polynomial properties. Our protocols for verifiable delegation of computation are aimed to the efficient evaluation of polynomials, working on schemes where the polynomial and/or the input are kept secret to the server. Our proposal for remote electronic voting allows the verification of vote well-formation upon reception at the voting server, with little overhead of computations for the voter.

1 Introduction

Delegation of computation has gained relevance over the last years due to the arise of mobile devices, such as smartphones, netbooks..., which cannot perform large computations themselves. The common solution then is to have the computations to be performed in the cloud. The two most important requirements in the delegation of computation are that the correctness of the result needs to be verifiable by the client, and that the work required for verifying it has to be substantially smaller than the amount required to perform the computation itself.

Specifically, we work on the verifiable delegation of computation of polynomials, since, as pointed out by Benabbas *et al.* in [2], by foccusing on specific functions we are able to provide more efficient protocols. They also provided the first scheme for polynomial delegation, and introduced the definition of pseudo-random functions with closed form efficiency (PRF with CFE). This latter definition was generalized and used by Fiore and Genaro in [5] to delegate, along with polynomial evaluation, matrix multiplication.

In this paper we present a set of efficient protocols that use PRF with CFE to allow a client to verifiably delegate the evalutation of (potentially large) polynomials. These protocols range from the standard setting, in which the client shares both the polynomial and the input point in the clear with the server, to settings where the client can hide the polynomial or the input point (or both). Most important, the security of our protocols is based on standard assumptions.

On the other hand, some of the polynomial properties we have used for constructing efficient protocols for delegation of computation schemes can also be used in an electronic voting scenario.

A common problem in remote electronic voting is how to check the validity of an encrypted vote without decrypting it, that is, how to check that a received ciphertext contains a valid vote option. Homomorphic tally schemes, where an aggregation of votes is decrypted using an homomorphic encryption scheme, to obtain the election results, are very common in electronic voting. Since votes are not decrypted individually, it is crucial to ensure that each recieved vote is well-formed in order for it not to distort the result. Even if votes are decrypted individually, it is desirable for the voting system to check the correctness of the vote contents upon reception, in order to increase confidence in the system. Needless to say, the proof of well-formation has to reveal no more information about the ciphertext beyond whether or not the vote is well-formed.

Although solutions to this problem already exist, we present a more efficient protocol for remote electronic voting based on polynomial commitments that allows a voting server to check the well-formation of an encrypted vote.

The paper is organized as follows: in section 2 some definitions and assumptions are presented, an introduction to verifiable computation is done in section 3, our proposal for delegating the computation of large polynomials is explained in section 4, the scheme for committing to polynomials from Kate *et al.* [7] is introduced in section 5 and, finally, in section 6 we show our proposal for a verifiable electronic voting scheme based on polynomial properties.

2 Definitions and assumptions

2.1 Pseudo-random functions with closed form efficiency

We adopt the definition given by Fiore *et al.* in [5], which generalizes the definition given by Benabbass *et al.* in [2].

2.1.1 Pseudo-random functions

A pseudo-random function (PRF) consists of two algorithms (PRF.KG, F). The key generation algorithm, PRF.KG, takes as input a seurity parameter λ and outputs a secret key K, along with some public parameters (pp) that specify the domain ($\mathcal{X}$) and range ($\mathcal{Y}$) of the function. Given an input $x \in \mathcal{X}$, $\mathrm{F}_K(x)$ uses the secret key to compute $y \in \mathcal{Y}$.

A PRF must satisfy the pseudo-randomness property: (PRF.KG, F) is ε-secure if, for every probabilistic polynomial time (PPT from now on) adversary $\mathcal{A}$:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{PRF}} = \left|\mathrm{Prob}\left[\mathcal{A}^{\mathrm{F}_K(\cdot)}(1^\lambda, \mathrm{pp}) = 1\right] - \mathrm{Prob}\left[\mathcal{A}^{R(\cdot)}(1^\lambda, \mathrm{pp}) = 1\right]\right| \leq \varepsilon$$

where $R(x)$ is a random function.

2.1.2 Closed Form Efficiency

Given an arbitrary computation $\mathtt{Comp}$ that takes as input l random values $R_1, \ldots, R_l \in \mathcal{Y}$ and m arbitrary values $\vec{x} = (x_1, \ldots, x_m)$, we assume that the best algorithm that computes $\mathtt{Comp}(R_1, \ldots, R_l, x_1, \ldots, x_m)$ takes a time T. Let $\vec{z} = (z_1, \ldots, z_l)$ be a set of arbitrary values in the domain $\mathcal{X}$ of F. We say that $\mathcal{PRF} =$ (PRF.KG,F) has CFE for $(\mathtt{Comp}, \vec{z})$ if there exists an algorithm $\mathrm{PRF.CFEval}_{\mathtt{Comp},\vec{z}}(K, x)$ such that:

$$\mathrm{PRF.CFEval}_{\mathtt{Comp},\vec{z}}(K, x) = \mathtt{Comp}(F_K(z_1), \ldots, F_K(z_l), x_1, \ldots, x_m)$$

and its running time is $\mathcal{O}(T)$.

In the case that the input of $\mathrm{F}_K(i)$ is $\{0, 1, 2, \ldots, l\}$ (i.e. , $z_i = i$), we omit the subscript $\vec{z}$.

2.2 One-way functions

Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function, we say that f is one-way if it is:

Easy to compute There exists an algorithm M_f that computes, in polynomial time, $f(x)\ \forall\ x \in \{0,1\}^*$.

Hard to invert Given a security parameter λ and a PPT adversary $\mathcal{A}$, there exists a negligible function ($\mathrm{neg}(\cdot)$) such that:

$$\mathrm{Prob}\left[x \leftarrow \{0,1\}^*, y = f(x) : \mathcal{A}(1^\lambda, y) \text{ find } x' \neq x \text{ such that } f(x') = y\right] \leq \mathrm{neg}(\lambda)$$

2.3 Additively homomorphic encryption schemes

Let $\langle \text{AH-KG}, \text{AH-Enc}, \text{AH-Dec} \rangle$ be a public key probabilistic encryption scheme, where the message space is an additive group and de ciphertext space a multiplicative one. We say that it is an additively homomorphic (AH) encryption scheme if, given $\gamma_1 = \text{AH-Enc}[m_1, r_1]$ and $\gamma_2 = \text{AH-Enc}[m_2, r_2]$, there exists r such that

$$\gamma_1 \gamma_2 = \text{AH-Enc}[m_1 + m_2, r]$$

2.4 Hardness assumptions

Consider a PPT algorithm $\text{RSAgen}(1^\lambda)$ that generates two factors, p and q, of equal length and prime except with probability negligible in λ ([1]). Let $N = pq$ with $(p, q) \leftarrow \text{RSAgen}(\lambda)$ and $\langle g \rangle = \mathbb{G}$ be a group of composite order N. Consider the subgroups $\langle g_p \rangle = \mathbb{G}_p$ and $\langle g_q \rangle = \mathbb{G}_p$ of $\mathbb{G}$ of order p and q, respectively.

Decisional subgroup membership problem: Given $g_{p0} \in \mathbb{G}_p$ and $g_0 \in \mathbb{G}$, we define the advantage of an adversay $\mathcal{A}$ in solving the decisional subgroup membership problem as

$$\text{Adv}_{\mathcal{A}}^{\text{DSM}} = |\text{Prob}\left[\mathcal{A}(g_p, \mathbb{G}, N, g_{p0}) = 1\right] - \text{Prob}\left[\mathcal{A}(g_p, \mathbb{G}, N, g_0) = 1\right]|$$

Computational quadratic residuosity problem: Consider $y \in_R \mathbb{Z}_N$, we define the advantage of an adversary $\mathcal{A}$ in solving the computational quadratic residuosity problem as

$$\text{Adv}_{\mathcal{A}}^{\text{CQR}} = \text{Prob}[x \leftarrow \mathcal{A}(1^\lambda, N, y) \ : \ x^2 = y \mod N]$$

Discrete logarithm problem: Consider now that $\mathbb{G}$ is a group of any order n, we define the advantage of an adversary $\mathcal{A}$ in solving the discrete logarithm problem as

$$\text{Adv}_{\mathcal{A}}^{\text{DL}} = \text{Prob}[r \leftarrow \mathcal{A}(1^\lambda, g, n, h) \ : \ g^r = h :]$$

t-Strong Diffie-Hellmann problem: Consider that $\mathbb{G}$ is now a group of prime order p. Given $\alpha \in_R \mathbb{Z}_p*$, and a $(t+1)$-tuple $\left\langle g, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^t} \right\rangle$, we define the advantage of an adversary $\mathcal{A}$ in solving the t-strong Diffie-Hellmann problem as

$$\text{Adv}_{\mathcal{A}}^{t\text{-SDH}} = \text{Prob}[g^{\alpha^{t+1}} \leftarrow \mathcal{A}(1^\lambda, g, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^t})]$$

We say that an assumption holds for some of this problems if, for every PPT adversary $\mathcal{A}$, its advantage in solving this problem is negligible in λ.

3 Verifiable computation

A verifiable computation scheme is a two-party protocol where (in the simplest setting and in a general way) a client chooses a function and an input, and gives them to a server. The latter is expected to evaluate the function and send back the result of the evaluation along with a "proof" that the evaluation is correct. Finally, the client verifies that the value and the proof match.

In other cases, though, it might be of interest to hide the function, the input, or both. In such cases, the client will give the server some values that will allow the server to perform some computation. The result of this computation, finally, will allow the client to obtain the desired value with few additional computation.

The main goal is the verification (and the extraction of the evaluation) to be efficient and, specifically, way more faster thant the function itself. The *amortized* model by Gennaro *et al.* [6] is adopted: for each function, the client is allowed to perform a one-time expensive computation effort to produce a public-private key pair. This keys will be used to efficiently verify (i.e., in linear time) the computation of the server in many inputs.

[1] From now on, RSAgen will denote such algorithm.

3.1 Algorithms

A verifiable computation scheme $\mathcal{VC} = (\textbf{KeyGen}, \textbf{ProbGen}, \textbf{Compute}, \textbf{VerifyComp}, \textbf{ExtractEval})$ consists of the following algorithms:

KeyGen$(1^\lambda, f) \rightarrow (\textbf{PK}, \textbf{SK})$ Based on the security parameter λ, the key generation algorithm generates a public-secret key value pair for the specified function f.

ProbGen$(\textbf{SK}, \alpha) \rightarrow (c_1, c_p)$ Using the secret key and based on the desired input α, the problem generation algorithm will produce two values that will be sent to the server and that will allow it to perform the requested computation.

Comp$(\textbf{PK}, c_1, c_p) \rightarrow (\gamma, t)$ Using the public key and the output of the problem generation algorithm, the server performs the requested computation (γ) and produces a proof (t) of the validity of γ.

VerifyComp$(\textbf{SK}, \gamma, t) \rightarrow \sigma_y$ The client accepts γ if t is a valid proof for γ and α.

ExtractEval$(\textbf{SK}, \sigma_y) \rightarrow y$ If γ has been correctly verified, the client extracts the desired value (y) from σ_y.

3.2 Properties

A verifiable computation scheme must satisfy the following properties:

3.2.1 Correctness

Intuitively, a verifiable computation scheme is correct if the problem generation algorithm allows an honest server to compute values that will be correctly verified by the client, and that will allow the client to obtain the value of the function in the desired input. Formally:

Definition 3.1. We say that a verifiable computation scheme, $\mathcal{VC}$, is correct in $\mathcal{F}$ if, for each function $f \in \mathcal{F}$, the key generation algorithm produces two keys $(\text{PK}, \text{SK}) \leftarrow \textbf{KeyGen}(\lambda, f)$ such that, if $(c_1, c_p) \leftarrow \textbf{ProbGen}(\text{SK}, x)$, $(\gamma, t) \leftarrow \textbf{Comp}(\text{PK}, c_1, c_p)$, $\sigma_y \leftarrow \textbf{VerifyComp}(\text{SK}, \gamma, t)$ and $y \leftarrow \textbf{ExtractEval}(\text{SK}, \sigma_y)$, then:

1. $\sigma_y \neq \perp$, and
2. $y = f(x)$

3.2.2 Security

Also intuitively, a verifiable computation scheme is secure if a malicious server can not persuade the verification algorithm to accept an incorrect computation. It is, given a function f and an input x, a malicious server will not be able to convince the verification algorithm to output $\sigma_y \neq \perp$ such that $y \leftarrow \textbf{ExtractEval}(\text{SK}, \sigma_y)$ with $y \neq f(x)$. This intuition is formalized using the following experiment:

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathbf{Verif}}(\mathcal{VC}, 1^\lambda, f)$
  $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathbf{KeyGen}(1^\lambda, f)$
  for $j = 1, 2, \ldots, l = \mathrm{poly}(\lambda)$ **do**
    $x_j \leftarrow \mathcal{A}(\mathrm{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots j-1})$
    $(c_{1,j}, c_{p,j}) \leftarrow \mathbf{ProbGen}(\mathrm{SK}, x_j)$
    $(\tilde{\gamma}_j, \tilde{t}_j) \leftarrow \mathcal{A}(\mathrm{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots j-1}, x_j, c_{1,j}, c_{p,j})$
    $\sigma_{y,j} \leftarrow \mathbf{VerifyComp}(\mathrm{SK}, \gamma_j, t_j)$
  end for
  $x \leftarrow \mathcal{A}(\mathrm{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots l})$
  $(c_1, c_p) \leftarrow \mathbf{ProbGen}(\mathrm{SK}, x)$
  $(\tilde{\gamma}, \tilde{t}) \leftarrow \mathcal{A}(\mathrm{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots l}, x, c_1, c_p)$
  $\sigma_y \leftarrow \mathbf{VerifyComp}(\mathrm{SK}, \tilde{\gamma}, \tilde{t})$
  if $\sigma_y = \perp$ **then**
    return 0
  else
    $y \leftarrow \mathbf{ExtractEval}(\mathrm{SK}, \sigma_y)$
    if $y = f(x)$ **then**
      return 0
    else
      return 1
    end if
  end if

Definition 3.2. A verifiable computation scheme is secure in $\mathcal{F}$ if, for each function $f \in \mathcal{F}$ and for every PPT adversary $\mathcal{A}$:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{Verif}} = \mathrm{Prob}[\mathrm{Exp}_{\mathcal{A}}^{\mathrm{Verif}}(\mathcal{VC}, 1^\lambda, f) = 1] \leq \mathrm{neg}(\lambda)$$

3.2.3 Efficiency

Intuitively, a verifiable computation scheme is efficient if the time needed to generate the problem, verify the computation and extract the evaluation is much lower than the time needed to compute the function itself. Formally:

Definition 3.3. A verifiable computation scheme $\mathcal{VC}$ is efficient if, for every $x \in \mathrm{Domain}(f)$ and for every (γ, t), the needed time to compute $(c_1, c_p) \leftarrow \mathbf{ProbGen}(\mathrm{SK}, x)$ plus the time needed to verify $\mathbf{VerifyComp}(\mathrm{SK}, \gamma, t)$ and the needed time to extract $\mathbf{ExtractEval}(\mathrm{SK}, \sigma_y)$ is $\mathcal{O}(T)$, where T is the time needed to compute $f(x)$.

Notice that, following the example in Gennaro *et al.* [6], the time needed to generate the keys is not included.

4 Our verifiable delegation schemes for polynomials

A set of verifiable computation schemes to delegate the evaluation of polynomials based on the factorization assumption are described. The first is the "normal" (i.e., with both the polynomial and the function in clear) scheme, the second allows the user to hide the input in front of the server, and the third one is intended to allow the user to hide the polynomial. Finally, a fourth scheme can be obtained by mixing the second and the third ones to allow the client to hide both the input and the polynomial.

4.1 Polynomial delegation scheme

In this section we define a new protocol for polynomial delegation which is secure under the decisional subgroup membership assumption.

4.1.1 New PRF with CFE for polynomial delegation

Let $N = pq$ ($(p, q) \leftarrow \text{RSAgen}(1^\lambda)$), $R(x)$ be a polynomial with coefficients $r_i \in \mathbb{Z}_N$ and $\langle g \rangle = \mathbb{G}$ a group of order N. We define the following operation:

$$\text{ProjectPoly}(z^{r_0}, z^{r_1}, \ldots, z^{r_d}, \alpha) = z^{qR(\alpha)}$$

where $z^{r_0}, z^{r_1}, \ldots, z^{r_d} \in \mathbb{G}$ are random values and $\alpha \in \mathbb{Z}_N$ is arbitrary.

The following is a description of a new PRF with CFE for $(\text{ProjectPoly}, \{0, 1, \ldots, d\})$ used in the polynomial delegation scheme:

PRF.KG$(1^\lambda, d)$**:** Generate $N = pq$, where $(p, q) \leftarrow \text{RSAgen}(1^\lambda)$, as well as a group description $\langle N, g, \mathbb{G} \rangle \leftarrow \mathcal{G}(1^\lambda)$.

Choose $d + 1$ values $r'_i \in_R \mathbb{Z}_N$, $m \in [0, d]$ and $z \in \mathbb{G}$.

Output $K = \{p, q, m, r'_0, \ldots, r'_d\}$ and $\text{pp} = \{z, N, g, \mathbb{G}\}$.

F$_K(i)$**:** Compute r_i as follows:

- If $i \neq m$, $r_i = pr'_i$.
- Otherwise, $r_i = r'_i$.

Output $\text{F}_K(i) = z^{r_i}$.

PRF.CFEval$_{\text{ProjectPoly}}(K, \alpha)$**:** $\text{PRF.CFEval}_{\text{ProjectPoly}}(K, \alpha) = z^{qr'_m \alpha^m}$.

The correctness of the CFE is easily seen bearing in mind that $z^{N=pq} = 1$ (since $z \in \mathbb{G}$ and $\text{ord}(\mathbb{G}) = N$):

$$z^{qR(\alpha)} = z^{q \sum_{i=0}^{d} r_i \alpha^i} = \prod_{i=0}^{d} z^{q r_i \alpha^i} = z^{q r'_m \alpha^m} \prod_{i=0, i \neq m}^{d} \cancel{z^{pq r'_i \alpha^i}} = z^{r'_m \alpha^m}$$

Theorem 4.1. *If the decisional subgroup membership assumption holds for RSAgen*(1^λ)*, then (PRF.KG* $(1^\lambda, d)$*,* $F_K(i)$*) is a pseudo-random function.*

Proof. In the following, we prove that

- If there exists a PPT algorithm $\mathcal{A}$ such that, given a value $x = z^a$ for any $z \in \mathbb{G}$, knows whether $a \equiv 0 \mod p$ (outputs 1) or $a \not\equiv 0 \mod p$ (outputs 0) with non-negligible probability; then
- There exists a PPT algorithm $\mathcal{B}$ that can solve the decisional subgroup membership problem with the same probability.

Notice that the difference between an element generated by $\text{F}_K(i)$ and an element generated by a random function is that the elements generated by $\text{F}_K(i)$ are of the form z^{pr}, while the others are not.

Formally: let $\mathcal{A}$ be a PPT algorithm such that

$$\text{Adv}_{\mathcal{A}}^{\text{PRF}} = \left| \text{Prob}\left[\mathcal{A}^{\text{F}_K(\cdot)}(1^\lambda, \text{pp}) = 1\right] - \text{Prob}\left[\mathcal{A}^{R(\cdot)}(1^\lambda, \text{pp}) = 1\right] \right| = \epsilon > \text{neg}(\lambda)$$

Then $\mathcal{B}$ could use $\mathcal{A}$ to solve the decisional subgroup membership problem with probability ϵ. On input x, $\mathcal{B}$ would just have to run $\mathcal{A}$ with input x and output the output of $\mathcal{A}$.

If $\langle g_p \rangle = \mathbb{G}_p$ and $\langle g_q \rangle = \mathbb{G}_q$, then they are of the form $g_p = g^q$ and $g_q = g^p$. Bearing that in mind, we have that:

- If $x \in \mathbb{G}_q$, x is of the form $x = g_q^s = g^{ps}$ for some $s \in \mathbb{Z}_N$.
- If $x \notin \mathbb{G}_q$, x is of the form $x = g_p^r g_q^s = g^{qr} g^{ps} = g^{qr+ps}$ for some $r, s \in \mathbb{Z}_N$.

Notice that in the first case $\mathcal{A}$ will return 1 since the exponent is $ps \equiv 0 \mod p$; and, in the second case, $\mathcal{A}$ will return 0 since the exponent is $(ps + rq) \not\equiv 0 \mod p$. Then:

$$\mathrm{Adv}_{\mathcal{B}}^{\mathrm{DSM}} = \epsilon > \mathrm{neg}(\lambda)$$

contradicting the subgroup membership assumtion. □

4.1.2 Protocol description

KeyGen$(1^\lambda, P(x))$ Let $P(x)$ be a d-degree polynomial with coefficients $p_i \in \mathbb{Z}_N$ $(i \in [0, d])$. Compute $(K, \mathrm{pp}) \leftarrow \mathrm{PRF.KG}(1^\lambda, d)$.

Compute $z_i = \mathrm{F}_K(i)$ for $i \in [0, d]$, and define $t_i = z_i z^{ap_i}$ with $a \in_R \mathbb{Z}_p$.

Define the secret key $\mathrm{SK} = \langle a, K, \perp, \perp \rangle$ and the public key $\mathrm{PK} = \langle \{p_i\}, \{t_i\}, \mathrm{pp}, \perp \rangle$.

Return (SK, PK).

ProbGen$(\mathbf{SK}, \alpha)$ Return $(c_1, c_p) = (\alpha, \perp)$

Comp$(\mathbf{PK}, c_1, c_p)$ The server performs the following computation:

1. $c_i = c_1^i$ for each $i \in [0, d]$
2. $\gamma = \sum_{i=0}^{d} p_i c_i$
3. $t = \prod_{i=0}^{d} t_i^{c_i}$

And returns (γ, t).

VerifyComp$(\mathbf{SK}, \gamma, t)$ The client verifies if

$$t^q \stackrel{?}{=} z^{a\gamma q} \mathrm{PRF.CFEval}_{\mathrm{ProjectPoly}}(K, \alpha)$$

If true, returns $\sigma_y = \gamma$. Otherwise, returns $\sigma_y = \perp$.

ExtractEval$(\mathbf{SK}, \sigma_y)$ If $\sigma_y \neq \perp$ return $y = \sigma_y$. Otherwise, return $\perp$.

4.1.3 Protocol analysis

CORRECTNESS. Notice that, if the client and the server have behaved in an honest way:

$$t^q = \left(\prod_{i=0}^{d} t_i^{c_i} \right)^q = \prod_{i=0}^{d} t_i^{q\alpha^i} = \prod_{i=0}^{d} (z^{ap_i} \mathrm{F}_K(i))^{q\alpha^i} = z^{aq \sum_{i=0}^{d} p_i \alpha^i} \prod_{i=0}^{d} \left(F_K(i)^{q\alpha^i} \right)$$

$$= z^{aqP(\alpha)} \prod_{i=0}^{d} z^{q \sum_{i=0}^{d} r_i \alpha^i} = z^{qaP(\alpha)} z^{qR(\alpha)} = z^{qaP(\alpha)} \mathrm{PRF.CFEval}_{\mathrm{ProjectPoly}}(K, \alpha)$$

and, consequently, γ will be correctly verified by **VerifyComp**(SK, γ, t).

Furthermore, $y = \sigma_y = \gamma = \sum_{i=0}^{d} p_i c_i = \sum_{i=0}^{d} p_i \alpha^i = P(\alpha)$ and, consequently, **ExtractEval**(SK, σ_y) will return the correct evaluation of $P(x)$ in α.

SECURITY. Here we prove that the protocol described is secure under the assumption that (PRF.KG,F) is a pseudo-random function.

Theorem 4.2. *If (PRF.KG,F) is a pseudo-random function, then the procotol described above is a secure delegation scheme given the definition in 3.2.2.*

Proof. The proof is split in two steps: we first shift to an information-theoretic setting, and then we apply information-theoretic arguments to obtain security. Formally, the proof proceeds as a sequence of games. For each game i, we denote as X_i the event in which adversary wins (it is, by convincing the adversary to verify a false output).

Game 0. The game 0 is the security experiment defined in 3.2.2 played with the adversary $\mathcal{A}$.

Game 1. The game 1 is the same as game 0, except that the verification is performed in a different way: instead of using PRF.CFEval, the verifier computes $z^{qR(\alpha)} = \left(\prod_{i=0}^{d} g_i^{\alpha^i}\right)^q$, where $g_i = \mathrm{F}_K(i)$.

Game 2. The game 2 is the same as game 1, except that instead of using $g_i = \mathrm{F}_K(i)$, g_i are choosen randomly in $\mathbb{G}$.

Claim 4.1. $\mathrm{Prob}[X_0] = \mathrm{Prob}[X_1]$.

Proof. The only difference betwen game 0 and game 1 is the way in which $z^{qR(\alpha)}$ is computed, which does not affects the probability of the adversary winning. □

Claim 4.2. $|\mathrm{Prob}[X_1] - \mathrm{Prob}[X_2]| \leq \epsilon_{\mathcal{PRF}}$

Proof. The proof is straightforward from the definition of pseudo-randomness. □

Before proceeding to prove that the probability of winning the game 2 is small, we state and prove a claim that will be useful. Consider the following oracle:

$$\mathcal{O}_{p,q,a,\{p_i\},\{r_i\}}(\{x_i\}, y, \hat{t}) = \begin{cases} 1, & \text{if } \hat{t} = ay + \sum_i r_i x_i \mod p \text{ and } y \neq \sum_i p_i x_i \mod N \\ 0, & \text{otherwise} \end{cases}$$

Consider the following experiment:

Experiment $\mathbf{Exp}_{\mathcal{A}}(1^\lambda, N, d)$
  $N = pq$
  Choose $a \in_R \mathbb{Z}_N$
  for $i = 0, \ldots, d$ **do**
    $p_i \leftarrow \mathcal{A}(N, p_0, \hat{t}_0 \ldots, p_{i-1}, \hat{t}_{i-1})$
    Choose $r_i \in_R \mathbb{Z}_N$
    $\hat{t}_i = ap_i + r_i$
  end for
  Run $\mathcal{A}^{\mathcal{O}_{p,q,a,\{p_i\},\{r_i\}}(\{x_i\},y,\hat{t})}(N, p_0, \hat{t}_0, \ldots, p_d, \hat{t}_d)$, $l = \mathrm{poly}(\lambda)$ times with the described oracle
  if any time $\mathcal{O}$ returns 1 **then**
    return 1
  else
    return 0
  end if

Claim 4.3. Let $\mathcal{A}$ be a computationally unbounded adversary, we define its advantage in the previous experiment as:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{Exp}} = \mathrm{Prob}[\mathrm{Exp}_{\mathcal{A}}(1^\lambda, N, d) = 1]$$

If $\mathcal{A}$ makes at most l queries to the oracle $\mathcal{O}$, then:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{Exp}} \leq \frac{l}{p}$$

Proof.

$$\begin{aligned}
\text{Prob}[\text{Exp}_{\mathcal{A}}(\lambda, N, d) = 1] &= \text{Prob}[\mathcal{O}_{p,q,a,\{p_i\},\{r_i\}}(\{x_i\}, y, \hat{t}) = 1 \text{ for any of the } l \text{ queries}] \\
&\leq l\text{Prob}[\mathcal{O}_{p,q,a,\{p_i\},\{r_i\}}(\{x_i\}, y, \hat{t}) = 1 \text{ in one query}] \\
&= l\text{Prob}[(\hat{t} = ay + \sum r_i x_i \mod p) \cap (y \neq \sum p_i x_i \mod N)] \\
&= l \sum_{\hat{y} \neq \sum p_i x_i \bmod N} \text{Prob}[y = \hat{y}]\text{Prob}[\hat{t} = ay + \sum r_i x_i \mod p | y = \hat{y}] \\
&= l(N-1)\frac{1}{N}\frac{q}{N} = \frac{l}{p}\left(1 - \frac{1}{N}\right) \leq \frac{l}{p}
\end{aligned}$$

□

Claim 4.4. $\text{Prob}[X_2] \leq \frac{l}{p}$

Proof. We prove that:

- If there exists an adversary $\mathcal{A}$ that is capable of winning in the game 2 with $\text{Prob}[X_2] > \frac{l}{p}$,
- Then there exists an adversary $\mathcal{B}$ with $\text{Adv}_{\mathcal{B}}^{\text{Exp}} > \frac{l}{p}$, which has been proven false.

Notice that we could write an algorithm $\mathcal{O}'(\{x_i\}, \gamma, t)$ that uses the already described oracle to simulate the behavior of the **VerifyEval** algorithm:

Algorithm $\mathcal{O}'(\{x_i\}, \gamma, t)$
  $\hat{t} = \log_z t$
  $\sigma \leftarrow \mathcal{O}_{p,q,a,\{p_i\},\{r_i\}}(\{x_i\}, \gamma, \hat{t})$
  if $\sigma = 1$ **then**
    return γ
  else
    if $\gamma = \sum_{i=o}^{d} p_i x_i$ **then**
      return γ
    else
      return $\perp$
    end if
  end if

Using this algorithm, $\mathcal{B}$ proceeds as follows:

Algorithm $\mathcal{B}(N, p_0, \hat{t}_0, \ldots, p_d, \hat{t}_d)$

$P(x) = \sum_{i=0}^{d} p_i x^i$
$(K, \text{pp}) \leftarrow \text{PRF.KG}(1^\lambda, d)$
$\gamma_i = p_i \ \forall i \in [0, d]$
$t_i = z^{\hat{t}_i} \ \forall i \in [0, d]$
$PK = (\{\gamma_i\}, \{t_i\}, \text{pp}, \perp)$
for $j = 1, 2, \ldots, l = \text{poly}(\lambda)$ **do**
  $x_j \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots,j-1})$
  $(c_{1,j}, c_{p,j}) \leftarrow (x_j, 0)$
  $(\tilde{\gamma}_j, \tilde{t}_j) \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots,j-1}, x_j, c_{1,j}, c_{p,j})$
  $\sigma_{y,j} \leftarrow \mathcal{O}'(\{x_j^i\}_{0 \le i \le d}, \tilde{\gamma}_j, \tilde{t}_j)$
end for
$x \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots,l})$
$(c_1, c_p) \leftarrow (x, 0)$
$(\tilde{\gamma}, \tilde{t}) \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots,l}, x, c_1, c_p)$
return $(\{x^i\}, \tilde{\gamma}, \log_z \tilde{t})$

Notice that the adversary $\mathcal{A}$ of the Claim 4.3 (here refered to as $\mathcal{B}$) is computationally unbounded and, consequently, is capable of computing discrete logarithms and evaluate the polynomial. Notice that if $\mathcal{A}$ wins in game 2, then $\mathcal{B}$ wins in the experiment **Exp** for sure. So we have that:

$$\text{Adv}_{\mathcal{B}}^{\text{Exp}} \geq \text{Prob}[X_2] > \frac{l}{p}$$

which contradicts the claim 4.3. □

Finally, the probability of an adversary winning in the original security experiment is

$$\text{Prob}[X_0] \leq \epsilon_{\mathcal{PRF}} + \frac{l}{p} = \text{neg}(\lambda)$$

being proved the security of the scheme. □

EFFICIENCY. The computational cost for delegating the evaluation of a polynomial (ignoring, as explained, the key generation step) is 5 products and 3 exponentiations, while the cost of computing the polynomial itself is d products and d exponentiations.

4.2 Polynomial delegation protocol with private input

In this section the previous scheme is modified to delegate the polynomial evaluation without revealing the input α. We first describe a one-way function that allows us to hide the input value, as well as an algorithm (inspired by [9]) that allows to compute sums of powers without knowing the summands.

4.2.1 One-way function to delegate polynomials with private input

Be $N = pq$, where $(p, q) \leftarrow \text{RSAgen}(1^\lambda)$, and $\alpha, \beta \in \mathbb{Z}_N$, we define the function $\text{F} : \mathbb{Z}_N \times \mathbb{Z}_N \to \mathbb{Z}_N \times \mathbb{Z}_N$ as:

$$\text{F}(\alpha, \beta) = (\alpha + \beta, \alpha\beta) \mod N$$

Theorem 4.3. *If the computational quadratic residuosity assumption holds for RSAgen*(1^λ)*, then the function* $\text{F}(\alpha, \beta)$ *is a one-way function given the definition in 2.2.*

Proof. We prove that $\mathrm{F}(\alpha, \beta)$ satisfies both properties required to one-way functions:

EASY TO COMPUTE. The functions implies only a sum and a product.

HARD TO INVERT. Be $N = pq$ $((p, q) \leftarrow \mathrm{RSAgen}(1^\lambda))$, $(\alpha, \beta) \in \mathbb{Z}_n$ and $(c_1, c_p) \leftarrow \mathrm{F}(\alpha, \beta)$, it holds that:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{Invert}} = \mathrm{Prob}\left[(\alpha', \beta') \leftarrow \mathcal{A}\left(1^\lambda, N, c_1, c_p\right) \ : \ (\alpha' + \beta' = c_1 \ \cap \ \alpha'\beta' = c_p)\right] \leq \mathrm{neg}(\lambda)$$

Proof. Following we prove that

- If there exists an adversary $\mathcal{A}$ such that $\mathrm{Adv}_{\mathcal{A}}^{\mathrm{Invert}} = \varepsilon > \mathrm{neg}(\lambda)$,
- Then there exists an adversary $\mathcal{B}$ that solves the computational quadratic residuosity problem with non-negligible probability.

$\mathcal{B}$, on input y, would just run $(\alpha, \beta) \leftarrow \mathcal{A}(1^\lambda, N, c_1 = 0, c_p = -y)$, and return $|\alpha|$. Notice that:

$$\left.\begin{array}{ll} c_1 = \alpha + \beta & = 0 \\ c_p = \alpha\beta & = -y \end{array}\right\} \Rightarrow \beta = -\alpha \Rightarrow -\alpha^2 = -y \Rightarrow |\alpha| = |\sqrt{y}|$$

So $\mathcal{A}$ will return $\sqrt{y}$ with probability ε and, consequently, $\mathcal{B}$ will solve the problem with probability $\varepsilon > \mathrm{neg}(\lambda)$, contradicting the computational quadratic residuosity assumption. □

It is proven, then, that $\mathrm{F}(\alpha, \beta)$ is a one-way function. □

4.2.2 Algorithm for computing sums of powers

Be $N = pq$, where $(p, q) \leftarrow \mathrm{RSAgen}(1^\lambda)$, and $\alpha, \beta \in \mathbb{Z}_N$, we define

$$c_i = \alpha^i + \beta^i \mod N \ \forall i \in \mathbb{Z}_0^+$$

Be $c_1 = \alpha + \beta$ and $c_{prod} = \alpha\beta$, we define the following algorithm that computes $\vec{c} = \{c_o, c_1, \ldots, c_d\}$:

Algorithm **ComputeSumsOfPowers**(c_1, c_{prod}, d)
  $\vec{c} = \{c_0 = 2, c_1 = c_1, c_2, \ldots, c_d\}$
  for $i = 2$, $i \leq d$ **do**
    if $i = \dot{2}$ **then**
      $c_i = c_{i/2}^2 - 2c_{prod}^{i/2}$
    else
      $c_i = c_{i-1}c_1 - c_{i-2}c_{prod}$
    end if
  end for
  return $\vec{c}$

4.2.3 Protocol description

KeyGen$(1^\lambda, P(x))$ Let $P(x)$ be a d-degree polynomial with coefficients $p_i \in \mathbb{Z}_N$ $(i \in [0, d])$, compute $(K, \mathrm{pp}) \leftarrow \mathrm{PRF.KG}(1^\lambda, d)$.

Compute $z_i = \mathrm{F}_K(i)$ for each $i \in [0, d]$, and define $t_i = z_i z^{ap_i}$ with $a \in_R \mathbb{Z}_p$.

Choose $\beta \in \mathbb{Z}_N$ and compute $P(\beta)$.

Define the secret key as $\mathrm{SK} = \langle a, K, \langle \beta, P(\beta) \rangle, \perp \rangle$ and the public key as $\mathrm{PK} = \langle \{p_i\}, \{t_i\}, \mathrm{pp}, \perp \rangle$.

Return (SK, PK).

ProbGen$(\mathbf{SK}, \alpha)$ Return $(c_1, c_p) = (\alpha + \beta, \alpha\beta)$

Comp$(\mathbf{PK}, c_1, c_p)$ The server computes:

1. $\vec{c} = \text{ComputeSumsOfPowers}(c_1, c_p, d)$
2. $\gamma = \sum_{i=0}^{d} p_i c_i$
3. $t = \prod_{i=0}^{d} t_i^{c_i}$

And returns (γ, t)

VerifyEval$(\mathbf{SK}, \gamma, t)$ The client verifies if

$$t^q \stackrel{?}{=} z^{a\gamma q}\text{PRF.CFEval}_{\text{ProjectPoly}}(K, \alpha)\text{PRF.CFEval}_{\text{ProjectPoly}}(K, \beta)$$

If true returns $\sigma_y = \gamma$. Otherwise returns $\sigma_y = \perp$.

ExtractEval$(\mathbf{SK}, y)$ If $\sigma_y \neq \perp$ return $y = \sigma_y - P(\beta)$. Otherwise, return $\perp$.

4.2.4 Protocol analysis

CORRECTNESS. Notice that, if the client and the server have had an honest behavior:

$$t^q = \left(\prod_{i=0}^{d} t_i^{c_i}\right)^q = \prod_{i=0}^{d} t_i^{q(\alpha^i+\beta^i)} = \prod_{i=0}^{d} (z^{ap_i}\text{F}_K(i))^{q(\alpha^i+\beta^i)} = z^{aq\sum_{i=0}^{d} p_i(\alpha^i+\beta^i)} \prod_{i=0}^{d} \left(F_K(i)^{q(\alpha^i+\beta^i)}\right) =$$

$$= z^{aq(P(\alpha)+P(\beta))} \prod_{i=0}^{d} z^{q\sum_{i=0}^{d} r_i(\alpha^i+\beta^i)} = z^{qa(P(\alpha)+P(\beta))} z^{q(R(\alpha)+R(\beta))} =$$

$$= z^{qa(P(\alpha)+P(\beta))}\text{PRF.CFEval}_{\text{ProjectPoly}}(K, \alpha)\text{PRF.CFEval}_{\text{ProjectPoly}}(K, \beta)$$

and γ will be correctly verified by **VerifyEval**(SK, γ, t).

Furthermore, because $y = \sigma_y - P(\beta) = \gamma - P(\beta) = \sum_{i=0}^{d} p_i c_i - P(\beta) = \sum_{i=0}^{d} p_i(\alpha^i + \beta^i) - P(\beta) = P(\alpha) + P(\beta) - P(\beta) = P(\alpha)$, **ExtractEval**$(\text{SK}, y)$ will return the evaluation of $P(x)$ in α.

SECURITY. We prove that the described protocol is secure under the assumption that (PRF.KG,F) is a pseudo-random function.

Theorem 4.4. *If (PRF.KG,F) is a pseudo-random function, then the protocol described above is a secure verifiable delegation scheme according to the definition given in 3.2.2.*

Proof. The proof proceeds identically to the one in 4.1.3, except for some differences in the algorithm $\mathcal{B}$:

Algorithm $\mathcal{B}(N, p_0, \hat{t}_0, \ldots, p_d, \hat{t}_d)$

$P(x) = \sum_{i=0}^{d} p_i x^i$
Choose $\beta \in \mathbb{Z}_N$
$P(\beta) = \sum_{i=0}^{d} p_i \beta^i$
$(K, \text{pp}) \leftarrow \text{PRF.KG}(1^\lambda, d)$
$\gamma_i = p_i \ \forall i \in [0, d]$
$t_i = z^{\hat{t}_i} \ \forall i \in [0, d]$
$\text{PK} = (\{\gamma_i\}, \{t_i\}, \text{pp}, \perp)$
for $j = 1, 2, \ldots, l = \text{poly}(\lambda)$ **do**
  $x_j \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots j-1})$
  $(c_{1,j}, c_{p,j}) \leftarrow (x_j + \beta, x_j \beta)$
  $(\tilde{\gamma}_j, \tilde{t}_j) \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots j-1}, x_j, c_{1,j}, c_{p,j})$
  $\sigma_{y,j} \leftarrow \mathcal{O}'(\{x_j^i + \beta^i\}_{0 \le i \le d}, \tilde{\gamma}_j, \tilde{t}_j)$
end for
$x \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots l})$
$(c_1, c_p) \leftarrow (x + \beta, x\beta)$
$(\tilde{\gamma}, \tilde{t}) \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots l}, x, c_1, c_p)$
return $(\{x^i + \beta^i\}, \tilde{\gamma}, \log_z \tilde{t})$

□

EFFICIENCY. The computational cost for delegating the evaluation of the polynomial is of 6 products and 3 exponentiations, while the cost of computing the polynomial itself is of d products and d exponentiations.

PRIVACY OF α. α remains unknown to the server if the quadratic residuosity assumption holds.

Theorem 4.5. *If the computational quadratic residuosity assumption holds for RSAgen(1^λ), the probability of a PPT adversary to compute (α, β) knowing $(\alpha + \beta, \alpha\beta)$ is negligible.*

Proof. The proof is straightforward from the proof of one-wayness given in 4.2.1. □

4.3 Polynomial delegation protocol with private polynomial

In this section the scheme defined in 4.1 is adapted to achieve polynomial delegation without revealing the polynomial.

4.3.1 Protocol description

KeyGen$(1^\lambda, P(x))$ Let $P(x)$ be a d-degree polynomial with coefficients $p_i \in \mathbb{Z}_N$ $(i \in [0, d])$, compute $(K, \text{pp}) \leftarrow \text{PRF.KG}(1^\lambda, d)$.

Let ε be an additively homomorphic (AH) encryption scheme as described in Section 2.3 with ciphertext space equal to the range of F, compute $(\text{sk}, \text{pk}) = \text{AH-KG}(\lambda)$.

Compute, for each $i \in [0, d]$, $\gamma_i = \text{AH-Enc}_{\text{pk}}[p_i]$, $z_i = F_K(i)$ and $t_i = \gamma_i^a z_i$.

Define the secret key as $\text{SK} = \langle a, K, \perp, \text{sk} \rangle$ and the public key as $\text{PK} = \langle \{\gamma_i\}, \{t_i\}, \text{pp}, \text{pk} \rangle$.

Return (SK, PK).

ProbGen$(\mathbf{SK}, \alpha)$ Return $(c_1, c_p) = (\alpha, \perp)$

Comp$(\mathbf{PK}, c_1, c_p)$ The server computes:

1. $c_i = c_1^i$

2. $\gamma = \prod_{i=0}^{d} \gamma_i^{c_i}$
3. $t = \prod_{i=0}^{d} t_i^{c_i}$

and returns (γ, t).

VerifyEval$(\mathbf{SK}, \gamma, t)$ The client verifies if

$$t^q \stackrel{?}{=} \gamma^{aq}\text{PRF.CFEval}_{\text{ProjectPoly}}(K, \alpha)$$

If true returns $\sigma_y = \gamma$. Otherwise returns $\sigma_y = \perp$.

ExtractEval$(\mathbf{SK}, \sigma_y)$ If $\sigma_y \neq \perp$ return $y = \text{AH-Dec}_{\text{sk}}[\sigma_y]$. Otherwise return $\perp$.

4.3.2 Protocol analysis

CORRECTNESS. Notice that, if the client and the server have behaved honestly:

$$t^q = \left(\prod_{i=0}^{d} t_i^{c_i}\right)^q = \prod_{i=0}^{d} (\gamma_i^a F_K(i))^{qc_i} = \prod_{i=0}^{d} \gamma_i^{aqc_i} F_K(i)^{qc_i}$$
$$= \left(\prod_{i=0}^{d} \gamma_i^{c_i}\right)^{aq} z^{q\sum_{i=0}^{d} r_i \alpha^i} = \gamma^{aq} z^{R(\alpha)} = \gamma^{aq}\text{PRF.CFEval}_{\text{ProjectPoly}}(K, \alpha)$$

and γ will be accepted by **VerifyEval**(SK, γ, t).

Furthermore, knowing that:

$$y = \text{AH-Dec}_{\text{sk}}[\sigma_y] = \text{AH-Dec}_{\text{sk}}[\gamma] = \text{AH-Dec}_{\text{sk}}\left[\prod_{i=0}^{d} \gamma_i^{c_i}\right]$$
$$= \sum_{i=0}^{d} \text{AH-Dec}_{\text{sk}}\left[\gamma_i^{c_i}\right] = \sum_{i=0}^{d} \text{AH-Dec}_{\text{sk}}\left[\gamma_i\right] \alpha^i = \sum_{i=0}^{d} p_i \alpha^i = P(\alpha)$$

we see that **ExtractEval**(SK, σ_y) will return the correct evaluation of $P(x)$ in α.

SECURITY. We prove that the described protocol is secure under the assumption that (PRF.KG,F) is a pseudo-random function.

Theorem 4.6. *If (PRF.KG,F) is a pseudo-random function, then the protocol described above is a secure verifiable delegation scheme as defined in 3.2.2.*

Proof. Notice that the verification is:

$$t^q \stackrel{?}{=} \gamma^{aq}\text{PRF.CFEval}_{\text{ProjectPoly}}(K, \alpha) = \gamma^{aq} z^{qR(\alpha)}$$

We can find analogy with the scheme in 4.1.3 by using:

$$\hat{p_i} = \log_z \gamma_i,\ \hat{y} = \log_z \gamma$$

We can now re-write the verification as:

$$t^q \stackrel{?}{=} z^{qa\hat{y}} z^{qR(\alpha)}$$

So the proof proceeds identically as in 4.1.3, except that $\mathcal{B}$ now proceeds as follows:

Algorithm $\mathcal{B}(N, p_0, \hat{t}_0, \ldots, p_d, \hat{t}_d)$
$P(x) = \sum_{i=0}^{d} p_i x^i$
$(K, \text{pp}) \leftarrow \text{PRF.KG}(1^\lambda, d)$
$(\text{sk}, \text{pk}) \leftarrow \text{AH-KG}(1^\lambda)$
$\gamma_i = z^{p_i}\ \forall i \in [0, d]$
$t_i = z^{\hat{t}_i}\ \forall i \in [0, d]$
$PK = (\{\gamma_i\}, \{t_i\}, \text{pp}, \perp)$
for $j = 1, 2, \ldots, l = \text{poly}(\lambda)$ **do**
$x_j \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots j-1})$
$(c_{1,j}, c_{p,j}) \leftarrow (x_j, 0)$
$(\tilde{\gamma}_j, \tilde{t}_j) \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots j-1}, x_j c_{1,j}, c_{p,j})$
$\sigma_{y,j} \leftarrow \mathcal{O}'(\{x_j^i\}_{0 \le i \le d}, \log_z \tilde{\gamma}_j, \tilde{t}_j)$
end for
$x \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots l})$
$(c_1, c_p) \leftarrow (x, 0)$
$(\tilde{\gamma}, \tilde{t}) \leftarrow \mathcal{A}(\text{PK}, \{x_i, c_{1,i}, c_{p,i}, \tilde{\gamma}_i, \tilde{t}_i, \sigma_{y,i}\}_{i=1\ldots l}, x, c_1, c_p)$
return $(\{x^i\}, \log_z \tilde{\gamma}, \log_z \tilde{t})$

□

EFFICIENCY. The computational cost of delegating the polynomial evaluation is of 4 products, 3 exponentiacions and 1 (de)cypher, and the cost of computing the evaluation itself is of d products and d exponentiations.

PRIVACY OF $P(X)$. If the cypher scheme ε is secure, the adversary cannot obtain any information about the coefficients of the polynomial.

5 Commitment to polynomials

A commitment scheme is such that a committer publishes a value called the *commitment*, which is connected, or bounded, to a message that is not revealed until later on, when the committer opens the commitment and reveals the hidden message in such a way that a verifier can check that the message is consistent with the commitment. A commitment scheme has the following properties:

- Binding: a commitment cannot be opened to an arbitrary message, but to the message to which it is initially related.
- Hiding: the value of the commitment does not provide information about the value of the committed message.

Verifiable secret sharing schemes use a polynomial to share a secret value among several participants, so that each share is the evaluation of that polynomial at some point. In order to preserve the secrecy of the value to share, the polynomial coefficients must remain secret. However, a verifiable secret sharing scheme requires the evaluation of the polynomial to be verifiable (at least) by the participants in the protocol. Kate *et al.* [7] propose a commitment scheme that allows a committer to commit to a polynomial, while different evaluations of this polynomial can be computed, by the participants in the protocol, without having to open the commitment and disclose the polynomial coefficients.

The authors propose an efficient scheme to commit to polynomials $P(x) \in \text{Z}_p[x]$ over a bilinear pairing group where:

- The size of the commitment is constant with the degree of the polynomial.

- The committer can efficiently open the commitment to any evaluation $P(i)$ using an auxiliar element called *witness*, allowing a verifier to confirm that $P(i)$ is the evaluation at i of the polynomial $P(x)$.
- The hiding property of the scheme is based on the DL assumption.
- The binding property is proven under the t-SDH assumption.

A stronger commitment scheme based on Pedersen commitments, unconditional hiding and computational binding under the t-SDH is also proposed by the authors. For the sake of clarity we will present the basic commitment scheme, but this can be easily transformed to be unconditionally hiding using the Pedersen commitment-based scheme explained in the original article.

5.1 Algorithms

A polynomial commitment scheme *PolyCommit* = (**Setup**, **Commit**, **Open**, **VerifyPoly**, **CreateWitness**, **VerifyEval**) is composed by the following algorithms:

Setup$(1^\lambda, t) \rightarrow (\mathbf{SK}, \mathbf{PK})$ Based on the security parameter λ, the setup algorithm chooses two groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order p for which a symmetric bilinear pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ exists. The setup algorithm chooses a generator $g \in_R \mathbb{G}$, a secret key SK $= \alpha \in_R \mathbb{Z}_p^*$ and a public key $PK = (g, g^\alpha, \ldots, g^{\alpha^t})$.

Commit$(\mathbf{PK}, P(x)) \rightarrow \mathcal{C}$ The commit algorithm computes the commitment $\mathcal{C} = g^{P(\alpha)} \in \mathbb{G}$ for the polynomial $P(x) \in \mathbb{Z}_p[X]$ of degree at most t. For a polynomial $P(x) = \sum_{j=0}^{deg(P)} p_j x^j$ the algorithm computes $\mathcal{C} = \prod_{j=0}^{deg(P)} (g^{\alpha^j})^{p_j}$.

Open$(\mathbf{PK}, \mathcal{C}) \rightarrow P(x)$ The open algorithm outputs the committed polynomial $P(x)$.

VerifyPoly$(\mathbf{PK}, \mathcal{C}, P(x)) \rightarrow (1/\perp)$ The verification algorithm verifies that $\mathcal{C} = g^{P(\alpha)}$, outputs 1 if verification succeeds and $\perp$ otherwise.

CreateWitness$(\mathbf{PK}, P(x), i) \rightarrow (i, P(i), \omega_i)$ The algorithm computes $w_i(x) = \frac{P(x)-P(i)}{(x-i)}$ and the witness $\omega_i = g^{w_i(\alpha)}$ computed in a similar way than $\mathcal{C}$.

VerifyEval$(\mathbf{PK}, \mathcal{C}, i, P(i), \omega_i) \rightarrow (1/\perp)$ The verification algorithm verifies that $P(i)$ is the evaluation of the polynomial $P(x)$ committed in $\mathcal{C}$ for the value i. If $e(\mathcal{C}, g) = e(\omega_i, g^\alpha / g^i) \cdot e(g, g^{P(i)})$ the algorithm outputs 1, $\perp$ otherwise.

6 Electronic voting

In a remote electronic voting scenario voters cast votes, containing encrypted voting options, to a voting service that stores them in a digital ballot box until the election ends. After that, a decryption service decrypts the votes, obtaining the selected voting options that are counted to obtain the election results.

Depending on the nature of the voting protocol, an anonymization process such as a Mix-Net (first in [3]) is used to break the link between the voter identities and votes prior to decryption. In other voting protocols, such as [4], the homomorphic properties of the algorithm for encrypting the voting options are used to decrypt the result of the operation of a set of votes, obtaining the aggregation of voting options instead of individual cleartext votes. This process is commonly called homomorphic tally. Traditionally, homomorphic tally schemes need the encrypted votes to be verified to contain only valid options upon reception, in order to prevent a voter to distort the result of the election with a misformed vote. However, increasingly, the schemes using Mix-Nets, where votes are decrypted individually and such misformations can be detected, are also required to check the correctness of the

vote contents upon reception, in order to detect software bugs in the vote casting process or to prevent oppositors claiming that they have been able to cast a misformed vote, what could reduce the public confidence on the system.

Since votes remain encrypted until the end of the election, zero knowledge proofs are used by the voter to prove to the voting service that the vote is well-formed without disclosing it. Specifically, the voter produces OR-proofs that demonstrate, for each encrypted option in the cast vote, that it corresponds to one of the options valid in the election. Although some proposals ([1],[8]) reduce the original cost of the OR-proof, the cost is still proportional to the number of options available in the election, regardless the number of selections the voter can make.

This section presents a remote electronic voting protocol that allows a voting service to check that received votes are well-formed, with proofs that have computational cost proportional to the number of selections made by voters.

The main idea of the protocol is the following: the voter generates an encryption of the vote in such a way that the voting service receiving it can check that the vote contains the encryption of valid options in that election, without disclosing which are those options. In order to do that, the voter builds its vote as the commitment to two polynomials, the first one, $P(x)$, contains as roots the chosen voting options, and the second, $Q(x)$, contains as roots the non-chosen voting options. The voting service, at receiving the vote, checks that the product of these two polynomials is equal to the polynomial containing as roots all the voting options allowed in that election. After the voting period, the decryption service receives all the votes received by the voting service and extracts the voter selections from the commited polynomials. Standard procedures like digital signatures for providing authenticity and integrity to the exchanged messages, as well as tools for anonymizing votes before decryption such as Mix-Nets are ommited in this protocol.

The polynomial commitment scheme in Section 5 is used for constructing the commitment to polynomials $P(x)$ and $Q(x)$, in such a way that the product of both polynomials can be computed from their commitments. However, for the decryption service being able to extract the selected voting options contained as roots in the polynomial $P(x)$, the voter will send individual commitments to the coeficients of $P(x)$, rather than the commitment of the whole polynomial.

6.1 Algorithms

The voting protocol for an election with n possible choices and k possible selections consists of the following algorithms:

Config$(1^\lambda, k, n) \rightarrow (PK_c, PK_e, SK_e, \mathcal{V}, T(x),$ ***Selections***$)$ Generate a keypair $PK_c = (g, g^\alpha, \ldots, g^{\alpha^k})$, $SK_c = \alpha$, following the setup for commitment scheme of Section 5. SK_c is securely deleted. Generate a second keypair PK_e, SK_e for a public key encryption scheme E_{PK_e}/D_{SK_e}. Set numerical values representing voting options $\mathcal{V} = \{v_1, \ldots, v_n\}$, $v_i \in \mathbb{Z}_p^*$, and construct $T(x) = \prod_{i \in \mathcal{V}}(x - v_i)$. Compute, for each subset $\mathcal{K}$ from $\mathcal{V}$ of k voting options, the polynomial $S(x) = \prod_{i \in \mathcal{K}}(x - v_i)$. Generate a *Selections* table mapping each subset $\mathcal{K}$ with the individual commitments of polynomial coefficients $C_{s_i} = g^{s_i \cdot \alpha^i}$.

CastVote$(PK_c, \mathcal{V}, PK_e) \rightarrow (\mathcal{C}_Q, \{\gamma_0, \ldots, \gamma_k\}, E_{PK_e}(a))$ Choose a set $\mathcal{S}$ from $\mathcal{V}$ of voting options and compose two polynomials, $P(x) = \prod_{i \in \mathcal{S}}(x - v_i)$, $Q(x) = \prod_{j \notin \mathcal{S}}(x - v_j)$. Select $a \in_R \mathbb{Z}_p^*$, then compute $\mathcal{C}_Q = g^{Q(\alpha)/a^k}$ using the commitment algorithm from Section 5. Commit to the individual coeficients of $P(x)$ as $\gamma_i = g^{a \cdot p_i \cdot \alpha^i}, \forall i \in \{0, \ldots, k\}$. Encrypt a and output vote $(\mathcal{C}_Q, \{\gamma_0, \ldots, \gamma_k\}, E_{PK_e}(a))$.

VerifyVote$(PK_c, T(x), \mathcal{C}_Q, \{\gamma_0, \ldots, \gamma_k\}, E_{PK_e}(a)) \rightarrow (1/\perp)$ Compute $g^{a^k \cdot P(\alpha)} = \prod_{i=0}^{k} \gamma_i$. Check that $e(g^{a^k \cdot P(\alpha)}, \mathcal{C}_Q) = e(g, g^{T(\alpha)})$. Output 1 if the verification succeeds, $\perp$ otherwise.

DecryptVote$(SK_e, \{\gamma_0, \ldots, \gamma_k\}, E_{PK_e}(a),$ ***Selections***$) \rightarrow$ **(Set** $\mathcal{K}$ **of** v_i**,** $i \in \mathcal{S}$**)** Obtain $a = D_{SK_e}$

$(E_{PK_e}(a))$ and recover individual values $g^{p_i \cdot \alpha^i}$. Check in the *Selections* table for correspondence with a set $\mathcal{K}$ of voting option values from $\mathcal{V}$ to obtain the original selections v_i made by the voter.

6.2 Protocol description

In the voting protocol, the participants: configuration entity, voter, voting service, decryption service and counting service follow the next steps:

- In the *Setup phase* the configuration entity runs the **Config**$(1^\lambda, k, n)$ algorithm to configure an election with n possible choices and k possible selections. The configuration entity publishes PK_c and PK_e, as well as the set of voting options $\mathcal{V}$. The voting service receives $T(x)$, and the decryption service receives SK_e and the *Selections* table.
- During the *Voting phase* the voters run the **CastVote**$(PK_c, \mathcal{V}, PK_e)$ algorithm to produce encrypted votes composed by the values $(\mathcal{C}_Q, \{\gamma_0, \ldots, \gamma_k\}, E_{PK_e}(a))$, which are sent to the voting service. When the voting service receives a vote, it runs the **VerifyVote**$(PK_c, T(x), \mathcal{C}_Q,$ $\{\gamma_0, \ldots, \gamma_k\}, E_{PK_e}(a))$ algorithm to ensure that the vote contents are correct (it contains a subset of k elements of $\mathcal{V}$). The vote is accepted if the algorithm outputs 1, rejected otherwise.
- After the election closes, in the *Decryption phase*, the decryption service runs the **DecryptVote** $(SK_e, \{\gamma_0, \ldots, \gamma_k\}, E_{PK_e}(a),$ *Selections*$)$ algorithm for all the votes, and for each one the subset $\mathcal{K}$ from $\mathcal{V}$ of voting options is obtained. After that, the counting service will count the times each voting option v_i is present in all subsets $\mathcal{K}$ in order to obtain the election results.

6.3 Protocol analysis

CORRECTNESS.

A vote which has been correctly formed is composed by the commitments of $P(x)$ and $Q(x)$, where $P(x) \cdot Q(x) = T(x)$. Therefore, the **VerifyVote** algorithm succeeds since $e(g^{a^k \cdot P(\alpha)}, g^{Q(\alpha)/a^k}) = e(g,g)^{P(\alpha) \cdot Q(\alpha)} = e(g,g)^{T(\alpha)} = e(g, g^{T(\alpha)})$.

The **DecryptVote** algorithm suceeds with overwhelming probability, since:

- $D_{SK_e}(E_{PK_e}(a)) = a$, $\mathcal{C}_{p_i} = g^{p_i \cdot \alpha^i} = \gamma_i^{1/a}$, $\forall i \in \{0, \ldots, k\}$.
- Table *Selections* maps groups of k voting options $v_i \in \mathcal{K}$, $\mathcal{K} \subset \mathcal{V}$ to the individual commitments $\mathcal{C}_{s_i}$ of the coefficients of a polynomial $S(x)$ with roots $v_i \in \mathcal{K}$. So, there's a chance that two polynomials map to the same set of individual commitments, meaning that back-conversion (from commitments to voting options) can be wrong. This probability is $\frac{k!}{p^k}$, which can be considered negligible since the generator g for making the commitments is chosen to be of a high order (p), while the maximum degree k of the polynomials to commit to is estimated to be a much smaller value.

SECURITY.

The hiding property of the polynomial commitment scheme $P(x) \rightarrow g^{P(\alpha)}$ is proven in [7] under the DL assumption. The binding property is proven under the SDH assumption.

However, in the scenario of an election where voters choose values within a set of public options, the commitment $\mathcal{C}_P$ can be used to break voter privacy. Therefore, the protocol uses randomized commitments $\mathcal{C}_P = g^{P(\alpha) \cdot b}$, $\mathcal{C}_Q = g^{Q(\alpha)/b}$, where $b = a^k$ and $a \in_R \mathbb{Z}_p^*$.

For breaking voter privacy, an attacker should be able to obtain the value b from $g^{P(\alpha) \cdot b}$, $g^{Q(\alpha)/b}$ or $E_{PK_e}(a)$. Assuming the encryption algorithm E is secure, being able to identify b, $g^{P(\alpha)}$ or $g^{Q(\alpha)}$ from the first two elements would be equivalent to be able to distinguish from the DH tuples $(g^{z_1}, g^{z_2}, g^{z_3}), (g^{z_1}, g^{z_2}, g^{z_1 \cdot z_2})$ which is not possible according the DH assumption.

EFFICIENCY.

The protocol for casting a vote, the content of which can be verified, has a cost proportional to the number of selections k and independent of the total of voting options in the election, n which is a significant improvement over current e-voting protocols.

Generation of the table *Selections* requires a much higher computational cost. However, this is done once, prior to the voting phase.

7 Conclusions

Taking advantage of polynomial properties, we have first presented a set of efficient protocols for a client verifiably delegating the evaluation of a large polynomial to a remote server. The delegation scheme allows the client to hide the polynomial and/or the input to the server, while being able to retrieve the result from the computation made by the remote server with a lower cost than computing the evaluation of the polynomial itself.

Secondly, we propose a scheme for electronic voting where the voter can demostrate to a remote voting service that her vote is well-formed, without having to disclose it. Our scheme is an improvement over existing proposals, since the cost of the proof is proportional to the number of selections the voter can do, instead of being proportional to the number of options available in the election.

References

[1] Mihir Bellare, Juan A. Garay, and Tal Rabin. Batch verification with applications to cryptography and checking. In Claudio L. Lucchesi and Arnaldo V. Moura, editors, *LATIN*, volume 1380 of *Lecture Notes in Computer Science*, pages 170–191. Springer, 1998.

[2] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer, 2011.

[3] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[4] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1997.

[5] Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. *IACR Cryptology ePrint Archive*, 2012:281, 2012.

[6] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[7] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[8] Kun Peng and Feng Bao. Batch zk proof and verification of or logic. In Moti Yung, Peng Liu, and Dongdai Lin, editors, *Inscrypt*, volume 5487 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2008.

[9] Jia Xu. Practically efficient verifiable delegation of polynomial and its applications. *IACR Cryptology ePrint Archive*, 2011:473, 2011.