Oblivious PRF on Committed Vector Inputs and Application to Deduplication of Encrypted Data

Jan Camenisch[1], Angelo De Caro[2], Esha Ghosh[3], and Alessandro Sorniotti[2]

[1] DFINITY Zurich Research Lab, Switzerland, jan@dfinity.org
[2] IBM Research, Zurich, Switzerland, {ADC,aso}@zurich.ibm.com
[3] Microsoft Research, Redmond, USA, esha.ghosh@microsoft.com

Abstract. Ensuring secure deduplication of encrypted data is a very active topic of research because deduplication is effective at reducing storage costs. Schemes supporting deduplication of encrypted data that are not vulnerable to content guessing attacks (such as Message Locked Encryption) have been proposed recently [Bellare *et al.* 2013, Li *et al.* 2015]. However in all these schemes, there is a key derivation phase that solely depends on a short hash of the data and not the data itself. Therefore, a file specific key can be obtained by anyone possessing the hash. Since hash values are usually not meant to be secret, a desired solution will be a more robust oblivious key generation protocol where file hashes need not be kept private. Motivated by this use-case, we propose a new primitive for oblivious pseudorandom function (OPRF) on committed vector inputs in the universal composable (UC) framework. We formalize this functionality as $\mathcal{F}_{\mathsf{OOPRF}}$, where OOPRF stands for *Ownership-based Oblivious PRF*. $\mathcal{F}_{\mathsf{OOPRF}}$ produces a unique random key on input a vector digest provided the client proves knowledge of a (parametrisable) number of random positions of the input vector.

To construct an *efficient* OOPRF protocol, we carefully combine a hiding vector commitment scheme, a variant of the PRF scheme of Dodis-Yampolskiy [Dodis *et al.* 2005] and a homomorphic encryption scheme glued together with concrete, efficient instantiations of proofs of knowledge. To the best of our knowledge, our work shows for the first time how these primitives can be combined in a secure, efficient and useful way. We also propose a new vector commitment scheme with constant sized public parameters but $(\log n)$ size witnesses where n is the length of the committed vector. This can be of independent interest.

1 Introduction

Cloud storage systems are becoming increasingly popular as a way to reduce costs while increasing availability and flexibility of storage. A promising technology that keeps the cost of cloud storage systems down is data deduplication, which can reduce up to 68 percent storage needs in standard file systems [25]. Data deduplication avoids storing multiple copies of the same data at the cloud storage. For example, if two clients upload the same file, the cloud server detects that, stores a single copy of the file and gives access to it to both clients.

However, if two clients locally encrypt their files with their individual keys, completely independent ciphertexts would result even if the underlying plaintext file is the same, thereby making deduplication impossible. A fundamental challenge in deduplicating encrypted files is the following: how can two mistrusting users obtain a common encryption key that depends on the content of a file they both own, without revealing anything about this fact, or about the file's content? Schemes that address this problem *and* are not vulnerable to content guessing or offline brute-force attacks have been proposed recently [22, 23]. But all these schemes rely on an oblivious key derivation phase, executed between a key server KS and a client C, whose input solely depends on a short hash of the file and not on the file itself. In these systems, a file-specific key will not only be revealed to the legitimate owners of the file but, crucially, to anyone knowing the hash of the file. This vulnerability will be particularly disastrous if a malicious party (modeling insider threat in cloud storage systems, like a malicious administrator) gets hold of a ciphertext and the hash of a file.

Hash values are usually not meant to be secret and are in fact openly used in multiple contexts, e.g. for checksumming, in standard deduplication protocols, in blockchain systems, and for authentication in Merkle trees. Note that the fundamental issue here is that the key generation solely depends on a *short hash* of the file, so getting this short hash is sufficient to get the key for the file through the oblivious protocol. This concern remains unaddressed even if domain separation [22] is used (i.e., domain specific salt is used for generating hash for the key server), since the oblivious key generation will still depend on the short hash. A desired solution will be a more robust oblivious key generation protocol where file hashes need not be kept private. In other words, any small leakage on a file, *should not be sufficient* to get the legitimate file specific key.

The obvious first attempt in achieving a robust oblivious key generation protocol is to add a proof of knowledge step in the key generation phase. The oblivious key generation phase is usually achieved using Oblivious Pseudorandom Function (OPRF). An OPRF [18] is a two-party protocol between Alice and Bob for securely computing a pseudorandom function $f_k(x)$ where Alice holds the key k and Bob wants to evaluate the function on input x. Despite its simplicity, OPRF has been shown to be a powerful primitive with application in multiple contexts [21, 18, 20] and in particular for secure-deduplication [22, 23] in cloud storage systems. Security of OPRF requires that Bob learns only $f_k(x)$ while Alice learns nothing from the interaction. However, the OPRF protocols in the literature [21, 20] can only handle large inputs with a considerable loss in efficiency when Bob is malicious. In particular, none of the OPRF functionalities in the literature can handle the following situation: Bob wants to evaluate the function on a short representation of his large input, while Alice wants Bob to prove knowledge of his large input, and not the short representation, *efficiently*, i.e., with communication complexity asymptotically smaller than the length of his input. We address this precise question here. Notice that this is exactly the question we are asking in the context of oblivious key generation for secure deduplication.

Is it possible to construct an OPRF protocol that can handle large input from a malicious party with communication complexity that is asymptotically strictly smaller than the size of the input?

In order to solve the conflict between the requirements expressed in the above question, we envision a protocol where the output of the OPRF still depends solely on the hash of the input, but that requires a user to prove knowledge of the pre-image of that hash in an *efficient way*, while retaining privacy. This implies that the system should enable efficient and compact proof of knowledge of the preimage of a hash without revealing anything about the hash or the preimage.

These multi-fold requirements naturally suggest combining a Proof-of-Ownership (PoW) [19] with an OPRF protocol. In a PoW protocol, ownership of a file is ascertained probabilistically by challenging the user to prove knowledge of certain blocks of the input file. However, by definition, a PoW scheme requires a deterministic hash of the file to be maintained at the server, and therefore, is stateful. Moreover, a PoW server, by definition, should be able to decide if two users possess the same file or not. Therefore, any PoW scheme falls short of our privacy goals where we do not want to reveal any information about the file in the proof-of-knowledge phase.

1.1 Our Result

We answer the question in the previous section in the affirmative by proposing a new OPRF primitive on committed vector inputs in the *universal composable* (UC) framework. We formalize this functionality as $\mathcal{F}_{\mathsf{OOPRF}}$ where OOPRF stands for *Ownership-based Oblivious PRF*. $\mathcal{F}_{\mathsf{OOPRF}}$ produces a unique random key on an input vector digest, only if the client proves knowledge of a (parametrisable) number of random positions of the input vector. By carefully tuning the number of positions to challenge the client on, bandwidth consumption can be reduced while ensuring that a malicious client can only cheat with negligible probability. We discuss how this tunable parameter should be set and how it affects the soundness error of the protocol. We further describe how to make our protocol more efficient in the weaker stand-alone security model.

Threat Model. The general setting of secure deduplication consists of three parties: a storage server (SS), a set of clients (C_i) who store their encrypted files on SS and a key server (KS) who aids the deduplication process by assisting the client to generate encryption keys that are unique for each file to be encrypted. In the first phase, the clients interact with the KS to get a key which is used to encrypt the input file such that the resulting ciphertext can be deduplicated. This is obtained as a result of the fact that a file encrypted with the same key will always produce the same ciphertext.

We will focus on the key generation phase that is executed between the KS and a client C_i. The clients are malicious and the KS is honest-but-curious in our threat model (we discuss how to tolerate a malicious KS in Section 4). This is a threat model that captures a wide range of realistic settings where a malicious client is in possession of ciphertexts (modeling an attacker hacking into the SS, or a malicious administrator of the SS with access to ciphertexts or an intelligence agency coercing the SS into releasing ciphertexts) and hashes of the files produced by an honest client. The malicious client can then try to obtain the decryption key for the file by fooling the KS. The KS typically models a cloud service provider that has no incentive in generating weak or incorrect keys for the files. A more realistic scenario is where the KS may stealthily deviate from the protocol to learn information about the client files. But we protect against any such information leakage. In other words, we protect the input privacy of the clients even against a malicious KS.

It is also easy to detect (with high probability) if KS is misbehaving in generating the key by sending the same file twice and observing if the same key is generated. Since KS generates the key obliviously, it will not be able to detect that it has received the same file. So, if KS is not faithfully generating the keys, with very high probability it will end up generating different keys and thus risk detection.

UC Security. With the increasing popularity of cloud platforms, significant effort went into developing customized solutions for various problems related to the security and privacy of outsourced data and computations. These protocols are not very modular by design and it is extremely challenging to compose them to achieve multiple security goals at once. In this work, we take an important step towards modular design by formalizing the security requirements in the *Universally Composable* (UC) framework which provides composable security guarantees.

Efficiency. Designing UC-secure protocols introduce some performance overheads. However, it is often trivial to optimize a UC-secure protocol (making it only secure in weaker models), whereas it is often extremely complex (if not outright infeasible) to demonstrate UC-security for a protocol that is only secure in weaker models. For achieving stand-alone security in our protocol, it is sufficient to instantiate all the zero-knowledge proofs of knowledge in our protocol with generalized Schnorr proofs and using the Fiat-Shamir heuristic, which are very efficient in practice.

Ideal Functionality. A naive attempt to define the ideal functionality, $\mathcal{F}_{\mathsf{OOPRF}}$, is the following. A (possibly malicious) client C hands in its entire input file to the functionality. If $\mathcal{F}_{\mathsf{OOPRF}}$ has not seen this file before, it generates a fresh random key, stores it with the file, and returns that key to the client. Otherwise, $\mathcal{F}_{\mathsf{OOPRF}}$ just returns the key it has stored for that file. Any realization of such a functionality would require communication between C and KS to be linear in length of C's input. This is because the simulator will need to extract on-line, the entire file from a malicious client, to be able to input it to the functionality. This defeats the compactness requirement we are looking for. To avoid this, we could let $\mathcal{F}_{\mathsf{OOPRF}}$ remember some *succinct representation* of each file and then allow the simulator to input just that representation and get the key from the functionality. However, this would let malicious clients get away safely with knowing the succinct representation only rather than the full file. This is precisely the security issue we are trying to overcome!

We envision a protocol where a client will just have to commit to the whole file (e.g., with a vector commitment) and then to prove that it knows sufficiently many blocks, it, can open sufficiently many random positions of the vector commitment. Vector commitments [13] allow a party to commit to a vector of messages in such a way that it can later provide a witness that proves that $x[i]$ is indeed the i-th value in the committed vector x.

To allow for such communication-efficient protocol realizations, we need to model inside the functionality, that a file is only provided partially. We do this by allowing the simulator to obtain keys from $\mathcal{F}_{\mathsf{OOPRF}}$ on input a *succinct representation* of a file together with *sufficiently many blocks* of the file, where "sufficiently many" is defined in terms of a security parameter t. $\mathcal{F}_{\mathsf{OOPRF}}$ will choose a fresh key for each representation of a file, store the key, the representation of the file and the provided blocks. When $\mathcal{F}_{\mathsf{OOPRF}}$ sees the same representation again, it will check whether the blocks on file for that representation are consistent with the newly provided blocks. The representation and blocks provided by the simulator will also have to be consistent with the *full file input* to $\mathcal{F}_{\mathsf{OOPRF}}$ by honest clients. So, $\mathcal{F}_{\mathsf{OOPRF}}$ will have to compute its own representation for full file input (it cannot ask the simulator as then the input of the honest client would no longer remain secret as we require). Thus, we need to provide $\mathcal{F}_{\mathsf{OOPRF}}$ with a function (vector commit) to compute this representation.

Notice that the guarantee that $\mathcal{F}_{\mathsf{OOPRF}}$ provides is that, when the input files are same, the client will get the same key. But the client cannot get the key by knowing a short representation for a large file; it has to show that it knows "sufficiently many" blocks of the file. This is significantly different from *Proof-of-Retrievability* [27] schemes where the guarantee is that the *entire* outsourced file is saved at all times (at the server).

Protocol. To construct a protocol that securely realizes $\mathcal{F}_{\mathsf{OOPRF}}$, we combine hiding vector commitments [13], a hiding and binding commitment scheme [26], a variant of the PRF scheme of Dodis-Yampolskiy [17, 21, 14], and a homomorphic encryption scheme [16, 8] together with *concrete, efficient instantiations of proofs of knowledge.* At a high level, the construction is designed as follows.

The PRF is obliviously evaluated on a succinct deterministic commitment to C's input $\boldsymbol{x}$, say s. We implement the oblivious PRF by leveraging the homomorphism of the encryption scheme. Now recall that C has to prove knowledge of random positions in the preimage of s *efficiently* and wants to preserve the confidentiality of its input. This can be addressed by using a randomized/hiding vector commitment. Still, the PRF needs to be evaluated on s, to ensure that the protocol always returns the same output given the same input. We provide an efficient proof of knowledge implementation that binds the randomized vector commitment with a commitment to s. Our protocol ensures that all these components can inter-operate efficiently. To the best of our knowledge, our work shows for the first time how these primitives can be combined in a secure, efficient and useful way.

As a subroutine of our protocol, we construct a new vector commitment (VC) scheme with *constant-sized public parameters* and $\log n$ size witnesses where n is the length of the committed vector. The scheme is based on the Merkle Hash Tree (MHT) based accumulator construction presented in [4]. Very recently [3] proposed a non-hiding MHT based VC with efficient batching of witnesses of position binding in groups of unknown order. Their batch openings only saves (asymptotically) when a few of the positions in the committed vector are set and all this positions are opened in a batch. This is incompatible with our requirement where a few positions are opened selectively. Moreover, their proofs require very expensive group operations in groups of unknown order.

1.2 Organization

The rest of the paper is organized as follows. In Section 2, we describe the cryptographic primitives. In Section 3 we describe the ideal functionality $\mathcal{F}_{\mathsf{OOPRF}}$ and in Section 4 we give a secure realization of $\mathcal{F}_{\mathsf{OOPRF}}$. One of the main building blocks of the protocol is vector commitment. In Section 5, we give an instantiation of VC and defer the second instantiation to Appendix D. Throughout the protocol and the VC implementations we use abstract PoK notation for the proofs of knowledge. In the appendix, we give the concrete implementations of all the PoK's (E).

2 Preliminaries

In this section we discuss the cryptographic primitives used in our protocol.

2.1 Proof Systems (PK)

By $\mathsf{PoK}\{(w) : \mathit{statement}(w)\}$ we denote a generic interactive zero-knowledge proof protocol of knowledge of a witness w such that the $\mathit{statement}(w)$ is true. A PoK system must fulfil *completeness, zero-knowledge and simulation-sound extractability*. A PoK system consists of the two protocols: $\mathsf{PK.Setup}$, $\mathsf{PK.Prove}$. On input a security parameter 1^λ, $\mathsf{PK.Setup}(1^\lambda)$ outputs $(\mathsf{par_{PK}})$. $\mathsf{PK.Prove}(\mathsf{par_{PK}}, \cdot)$ is an interactive protocol between prover and a verifier that $\mathit{statement}(w)$ is true. The additional input the prover holds is the witness w for the statement. Simulation-sound extractability for a PoK system requires the existence of an efficient algorithm $\mathcal{SE}$ that outputs $(\mathsf{par_{PK}}, \mathsf{td}_s, \mathsf{td}_e)$ such that $\mathsf{par_{PK}}$ is identically distributed to the $\mathsf{par_{PK}}$ generated by $\mathsf{PK.Setup}$ (td_e is the extraction trapdoor and td_s is the simulation trapdoor). When we need witnesses to be online-extractable, we make this explicit by writing $\mathsf{PoK}\{(\underline{w_1}, w_2) : \mathit{statement}(w_1, w_2)\}$ the proof of witnesses w_1 and w_2, where w_1 can be extracted.

For concrete realizations of PoK's, i.e., generalized Schnorr-signature proofs [7], we will use notation [9] such as $\mathsf{GSPK}\{(a, b, c) : y = g^a h^b \wedge \tilde{y} = \tilde{g}^a \tilde{h}^c\}$.

Whenever a witness needs to be on-line extractable, we will use verifiable encryption under a public key contained in the CRS. To allow for a proper assessment of our protocols, we will always spell these encryptions out (so there will not be any underlined witnesses in this notation). Finally, to make the 3-move generalized Schnorr-signature proofs concurrent zero-knowledge and simulations sound, one can use any of the standard generic techniques [15, 28], typically resulting in a 4-move protocol.

2.2 Commitment Scheme (CS)

We will instantiate CS with Pedersen commitment which satisfies correctness, hiding and binding properties. In addition to that, Pedersen commitments are homomorphic. We instantiate the commitment scheme in a composite order group to be compatible with the other primitives that we will be using [4].
$\underline{\mathsf{CS.Setup}(1^\lambda)\text{:}}$ The setup algorithm picks two λ bit safe primes p, q such that $\gcd(p-1, q-1, 7) = 1$ and sets $N = pq$ and sets message space and randomness space respectively as: $\mathcal{M} = \mathbb{Z}_N^*, \mathcal{R} = \mathbb{Z}_N^*$

Then, the algorithm picks a prime ρ such that $\rho = 2kN + 1$ where k is a small prime. Let $G = \langle \mathsf{G} \rangle = \langle \mathsf{H} \rangle$ be order-N subgroup of the group $\mathbb{Z}_\rho^*$ and G and H are two random generators of G, such that $\log_{\mathsf{H}} \mathsf{G}$ is unknown. Note that G is a cyclic subgroup of $\mathbb{Z}_\rho^*$ of order N and all the operations will happen modulo ρ (i.e., reduced $\bmod N$ in the exponent). Finally, the algorithm outputs public parameters $\mathsf{par} := (\rho, N, G, \mathsf{G}, \mathsf{H}, \mathcal{M}, \mathcal{R})$.
$\underline{\mathsf{CS.Commit}(\mathsf{par}, m, r)\text{:}}$ Compute $\mathsf{com} \leftarrow \mathsf{G}^m \mathsf{H}^r \mod \rho$. Output $(\mathsf{com}, \mathsf{open} = r)$
$\underline{\mathsf{CS.Verify}(\mathsf{par}, \mathsf{com}, m, \mathsf{open})\text{:}}$ Output 1 if $\mathsf{com} \leftarrow \mathsf{G}^m \mathsf{H}^{\mathsf{open}} \mod \rho$, 0 otherwise.

Theorem 1 ([26]). *The commitment scheme* CS *is information-theoretically hiding and binding under the Discrete Log assumption.*

2.3 Pseudorandom Function (PRF)

We will use the PRF scheme proposed in [21, 14] which is a variant of the PRF scheme of Dodis-Yampolskiy [17] based on the Boneh-Boyen unpredictable function [2], instantiated on a composite-order group instead of a prime-order group. This PRF was proven to be secure for a domain of arbitrary size based solely on subgroup hiding in [14]. The proof for the original PRF instantiated with prime-order groups only allows for a domain which is polynomial-sized in the security parameter. Notice that, for our application, it is crucial to have arbitrary size domain in order to disallow offline brute-force attack by a honest-but-curious KS. Here we recall the PRF definition and its security [14].
$\underline{\mathsf{PRF.Setup}(1^\lambda)\text{:}}$ On input the security parameter λ, the setup algorithm picks two λ-bit safe primes p, q and sets $N = pq$. Then, it generates groups $(N, \mathbb{G}, (\mathbb{G}_1, \mathbb{G}_2)) \leftarrow \mathcal{G}(1^\lambda)$, where $\mathbb{G}_1, \mathbb{G}_2$ are subgroups of $\mathbb{G}$.[4] Reasonable candidates for

[4] Notice that $\mathbb{G}_1, \mathbb{G}_2$ are not explicitly used in the construction, but are required from the security proof.

group $\mathbb{G}$ are composite-order elliptic curve groups *without* efficient pairings or the target group of a composite-order bilinear group. Finally, the setup algorithm picks $g \leftarrow \mathbb{G}$, sets the $\mathsf{D} = \mathsf{K} \leftarrow \mathbb{Z}_N^*, \mathsf{R} \leftarrow \mathbb{G}$, and output $\mathsf{par} = (N, \mathbb{G}, g, \mathsf{D}, \mathsf{K}, \mathsf{R})$.
$\underline{\mathsf{PRF.KeyGen}(1^\lambda, \mathsf{par})}$: On input the security and public parameters λ, par, the key generation algorithm picks $k \leftarrow \mathsf{K}$ and output k.
$\underline{\mathsf{PRF.Evaluate}(\mathsf{par}, k, m)}$: On input the public parameters par, key $k \in \mathsf{K}$ and input $m \in \mathsf{D}$, the evaluation algorithm does the following: If $\mathsf{gcd}((k+m), N) \neq 1$, then output $\perp$, else output $g^{\frac{1}{(k+m)}} \mod N \in \mathsf{R}$.

Theorem 2 ([14]). *For all $\lambda \in \mathbb{N}$, if subgroup hiding holds with respect to $\mathbb{G}$ for its subgroups $\mathbb{G}_1$ and $\mathbb{G}_2$, if $N = pq$ for distinct primes $p, q \in \Omega(2^{poly(\lambda)})$, and if $\mathbb{G}_1$ is a cyclic group of prime order, then the function family defined above is a pseudorandom function family.*

2.4 Homomorphic Encryption Scheme (HES)

Here we present the Projective Paillier Encryption scheme [16, 8]. This scheme preserves the homomorphic properties of Paillier encryption; however, unlike the original Paillier scheme, the scheme has a dense set of public-keys.
$\underline{\mathsf{HES.Setup}(1^\lambda)}$: On input the security parameter λ, the setup algorithm picks two λ bit safe primes p, q and set $N = pq$.[5] Then, generates a random element $g' \in (\mathbb{Z}_{N^2})$ and sets $\mathsf{g} := g'^{2N}$ and $\mathsf{h} := (1 + N \mod N^2) \in \mathbb{Z}_{N^2}^*$, a special element of order N. Finally, the algorithm outputs $\mathsf{par} := (N, \mathsf{g}, \mathsf{h})$.
$\underline{\mathsf{HES.KeyGen}(\mathsf{par})}$: On input the public parameters par, the key generation algorithm picks a random $t \in [N/4]$ and computes $\mathsf{epk} \leftarrow \mathsf{g}^t \mod N^2$. Finally, the algorithm outputs $(\mathsf{epk}, \mathsf{esk} := t)$.
$\underline{\mathsf{HES.Enc}(\mathsf{epk}, m)}$: On input the public key epk and message m, the encryption algorithm picks a random $r \in [N/4]$ and computes $\mathsf{u} \leftarrow \mathsf{g}^r \mod N^2; \mathsf{v} \leftarrow \mathsf{epk}^r \mathsf{h}^m \mod N^2$. Finally, the algorithm outputs ciphertext $\mathsf{ct} := (\mathsf{u}, \mathsf{v})$. We will sometime use the notation $[m]$ to mean the encryption of m.
$\underline{\mathsf{HES.Dec}(\mathsf{esk}, \mathsf{ct})}$: On input the secret key esk and ciphertext ct, the decryption algorithm computes $m' \leftarrow \mathsf{v}/\mathsf{u}^{\mathsf{esk}} \mod N^2$. If m' is of the form $(1 + Nm \mod N^2)$ for some $n \in [N]$, output m. Else output $\perp$.

Theorem 3 ([16]). *Under the Decision Composite Residuosity assumption, the Projective Paillier encryption scheme is semantically secure.*

2.5 Vector commitments (VC)

Vector commitments (VC) [13] allow one to commit to a vector of messages in such a way that it is later possible to open the commitment to one of the messages i.e, provide a witness that proves that x_i is indeed the i^{th} value in the committed vector $\boldsymbol{x}$. The size of the commitment and the opening are independent of the length of the vector. We relax the efficiency requirement of VC

[5] Algesheimer et al. describe how to generate such an N distributedly [1].

in our definition. Let n be the length of the committed vector. We require the size of the commitment to be independent from n, but the size of the opening should be asymptotically smaller than n, i.e., $o(n)$. A VC can either be *non-hiding/deterministic* (detVC) or *hiding/randomized* (randVC)[6]. For a detVC the only security requirement is *binding*. Informally, this property requires that once an adversary comes up with a VC, it should not be able to prove two different values with respect to the same position for that VC. For a randVC, the *hiding* is an additional security requirement. Informally, this requirement states that the VC should conceal the committed vector, i.e., an adversary should not be able to distinguish if a VC was created for a vector $\boldsymbol{x}$ or a vector $\boldsymbol{y}$, where $\boldsymbol{x} \neq \boldsymbol{y}$. For the formal definition of binding, refer to [13]. Hiding can be defined as for standard commitment.

Here we recall the primitives for a VC. Most of the inputs to the algorithms are common for a randVC and a detVC. The inputs that are needed exclusively for a randVC are highlighted.

$\underline{\mathsf{VC.Setup}(1^\lambda, n)}$: On input security parameter 1^λ and an upper bound n on the size of the vector, generate the parameters of commitment scheme par , which include a description of message space $\mathcal{M}$ and a description of randomness space $\mathcal{R}$. $\underline{\mathsf{VC.Commit}(\mathsf{par}, \boldsymbol{x}, r)}$: On input public parameters par, a vector $\boldsymbol{x} \in \mathcal{M}^l$, $(l \leq n)$ *and* $r \in \mathcal{R}$, the algorithm outputs a commitment com to $\boldsymbol{x}$. $\underline{\mathsf{VC.Prove}(\mathsf{par}, i, \boldsymbol{x}, r)}$: On input public parameters par, position index i, vector $\boldsymbol{x}$, *and* $r \in \mathcal{R}$, the algorithm generates a witness w for x_i and outputs (w, x_i). $\underline{\mathsf{VC.Verify}(\mathsf{par}, i, \mathsf{com}, w, x)}$: On input public parameters par, position index i, commitment com and witness w for x, the algorithm outputs 1 if w is a valid witness for x being at position i and 0 otherwise.

Below we define two new algorithms (VC.RandCommitment, VC.RandWitness). Informally, VC.RandCommitment allows to update a detVC to a randVC and VC.RandWitness allows to update a detVC witness to a randVC witness.

$\underline{\mathsf{VC.RandCommitment}(\mathsf{par}, \mathsf{com}, r)}$: On input public parameters par, a non-hiding commitment com and $r \in \mathcal{R}$, outputs a randVC com'.

$\underline{\mathsf{VC.RandWitness}(\mathsf{par}, \mathsf{com}, i, r, w)}$: On input public parameters par, a detVC witness w, a non-hiding commitment com and $r \in \mathcal{R}$, outputs a randVC witness w'.

UC Security: In Appendix A.1, we give a brief overview of UC security and direct the readers to [6, 10, 11] for more details.

[6] We will use the following terms interchangeably in the context of VC: non-hiding and deterministic, hiding and randomized.

3 Ideal Functionality for Ownership-based Oblivious PRF (OOPRF)

In this section we describe the ideal functionality $\mathcal{F}_{\mathsf{OOPRF}}$. As a warm-up, we start from a *bandwidth inefficient* version of $\mathcal{F}_{\mathsf{OOPRF}}$ denoted as $\mathcal{F}_{\mathcal{BI}-\mathsf{OOPRF}}$. The functionality is designed as follows:

1. $\mathcal{F}_{\mathcal{BI}-\mathsf{OOPRF}}$ receives input $\boldsymbol{x}$ from client C_i.
2. $\mathcal{F}_{\mathcal{BI}-\mathsf{OOPRF}}$ maintains a table to store the tuples $(\boldsymbol{x}, r_{\boldsymbol{x}})$ where $\boldsymbol{x}$ is the input from C_i and $r_{\boldsymbol{x}}$ is the *unique random key* that the functionality picks for $\boldsymbol{x}$.
3. For input $\boldsymbol{x}$ from C_i, if $\boldsymbol{x}$ is in the table, the functionality returns the corresponding $r_{\boldsymbol{x}}$ to C_i. Otherwise, it picks a fresh random key $r_{\boldsymbol{x}}$, stores $(\boldsymbol{x}, r_{\boldsymbol{x}})$ in its table and returns $r_{\boldsymbol{x}}$ to C_i.

Since the client has to hand in the entire set of blocks of a file[7] of length n to the functionality, any protocol that will achieve this functionality will be inefficient in terms of communication bandwidth. This is because the protocol will have to ensure that the entire file can be on-line extracted from a malicious client, which will amount to verifiably encrypting each file block. To overcome this, we are interested in a protocol where a client will just have to commit to the whole file (e.g., with a vector commitment) and then to prove that it knows sufficiently many blocks, i.e., can open sufficiently many random positions of the vector commitment. To allow for such a construction, we need to model inside the functionality, that a file is only provided partially. To this end, we will have to allow the simulator to obtain keys from the functionality on input a representation of a file together with sufficiently many blocks of the file, where "sufficiently many" is defined in terms of a security parameter t. The functionality will then choose a fresh key for each representation of a file, store the key, the representation of the file and the provided blocks. Furthermore, when in the future, the functionality sees the same representation again, it will check whether the blocks on file for that representation are consistent with the newly provided blocks. Of course, the representation and blocks provided by the simulator will also have to be consistent with the full file input to the functionality by honest clients. To this end, the functionality will have to compute its own representation (it cannot ask the simulator as then the input of the client would no longer remain secret as we require). Thus, we need to provide the functionality with a function to compute this representation. Thiss could either done by asking the simulator for this function in the setup phase or by parameterizing the functionality with this function. We chose the latter. Of course, the functionality will also have to enforce consistency between the blocks for the representations it computes itself and those it receives from the adversary/simulator.

Note that in $\mathcal{F}_{\mathsf{OOPRF}}$, both honest and malicious clients invoke the functionality with file $\boldsymbol{x}$. But, in case of malicious clients, the functionality generates the random key based on the input from the simulator and its internal state. The simulator's input is checked only against the stored internal state of the functionality, and not with respect to the input $\boldsymbol{x}$ with which a malicious client

[7] we use the word file and vector interchangeably

invokes the functionality. Thus, $\mathcal{F}_{\mathsf{OOPRF}}$ does not require the entire file to be on-line extracted from a malicious client.

```
Functionality $\mathcal{F}_{\mathsf{OOPRF}}$ (Parameterized with $\mathsf{detVC.Commit(par)}$)

Setup: Upon receiving $(\mathsf{Setup}, \mathsf{sid})$ from KS:
  1. Send $(\mathsf{Setup}, \mathsf{sid})$ to $\mathsf{Sim}$ and wait for $(\mathsf{Setup}, \mathsf{sid}, \mathsf{ok})$ from $\mathsf{Sim}$.
  2. Initialize an empty table $\mathsf{T}_{\mathsf{sid}}$.
  3. Store $(\mathsf{sid}, \mathsf{T}_{\mathsf{sid}})$.
  4. Output $(\mathsf{Setup}, \mathsf{sid})$ to KS.
Evaluate: Upon receiving input $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \boldsymbol{x})$ from $C_i$:
  1. Proceed only if $(\mathsf{sid}, \mathsf{T}_{\mathsf{sid}})$ are stored.
     If $C_i$ is honest: (a) Send $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{startEvaluate})$ to $\mathsf{Sim}$ and wait
             for $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{startEvaluateok})$ from $\mathsf{Sim}$.
        (b) Compute $s \leftarrow \mathsf{detVC.Commit}(\boldsymbol{x})$.
        (c) If $(s, \boldsymbol{x'}, r) \in \mathsf{T}_{\mathsf{sid}}$, for some $\boldsymbol{x'}$ and $r$, do the following:
              i. If the row corresponding to $s$ contains $\boldsymbol{x'} \neq \boldsymbol{x}$, set variable $\mathsf{out} \leftarrow$
                 $\perp$. Otherwise, $\mathsf{out} \leftarrow r$.
             ii. Output $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{out})$ to $C_i$.
        (d) Else, pick $r \leftarrow \mathcal{R}$, insert $(s, \boldsymbol{x}, r)$ in $\mathsf{T}_{\mathsf{sid}}$ and output
            $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, r)$ to $C_i$.
     If $C_i$ is malicious: (a) Send $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{startEvaluate})$ to $\mathsf{Sim}$ and
             wait for $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{startEvaluateok}, s, x[i_1], \ldots, x[i_t])$ from $\mathsf{Sim}$.
        (b) If $(s, \boldsymbol{x'}, r) \in \mathsf{T}_{\mathsf{sid}}$, for some $\boldsymbol{x'}$ and $r$, do the following:
              i. If $\boldsymbol{x'}$ contains $x'[i_1], \ldots, x'[i_t]$ for which the received
                 $x[i_1], \ldots, x[i_t]$ are unequal at least in one position, set $\mathsf{out} \leftarrow \perp$.
             ii. Else, update $\boldsymbol{x'}$ on positions $i_1 \ldots, i_t$ with values $x[i_1], \ldots, x[i_t]$,
                 respectively, and set $\mathsf{out} \leftarrow r$.
            iii. Output $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{out})$ to $C_i$.
        (c) Else, pick $r \leftarrow \mathcal{R}$, insert $(s, (x[i_1], \ldots, x[i_t]), r)$ in $\mathsf{T}_{\mathsf{sid}}$.
        (d) output $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, r)$ to $C_i$.
```

4 Secure realization of $\mathcal{F}_{\mathsf{OOPRF}}$

In this section we describe a protocol Π_{OOPRF} that securely realizes functionality $\mathcal{F}_{\mathsf{OOPRF}}$. We present the construction here and defer the proof of security to Appendix B.

4.1 Protocol Π_{OOPRF}

First we give the high level intuition behind our construction. The protocol is designed in the CRS model, so each party receives the public parameters of the scheme from a trusted party. KS additionally picks a key for a PRF. The protocol has two major building blocks, namely VC and PRF. Recall that the requirements for the key that KS will generate for C_i's input file were (1) the key should be random (2) it should be unique for a file and (3) the key should not be publicly computable. All these properties are provided if the file key is a PRF evaluation on a succinct *deterministic and binding* vector commitment to its input file $\boldsymbol{x}$. Let us denote this deterministic vector commitment as s.

The PRF evaluation has to be carried out obliviously as KS should learn no information about C_i's input. We implement the oblivious PRF evaluation protocol between KS (holding k) and C_i (holding s) for the PRF described in Section 2.3. To design this part of the protocol (Steps 7-13) we leverage the homomorphic encryption scheme that we described in Section 2.4. In order to tolerate malicious C_i's, we require that C_i commits to its input s, i.e., compute $\mathsf{com} = \mathsf{CS.Commit}(s, r)$ (where r is the randomness used to compute com using a standard commitment scheme as described in Section 2.2) and proves knowledge of its opening, r to KS before KS engages in computing the PRF ($\mathsf{PoK}\pi_{c2}$).

But committing to s is not sufficient; C_i has to prove knowledge of the preimage of s *efficiently*. This is where the properties of VC can be leveraged. A VC lets C_i prove knowledge of some random positions of the preimage. We utilize this property as follows: we let KS challenge C_i to prove knowledge of t random positions of its input, where t is much less than the length of $\boldsymbol{x}$. C_i can do this efficiently. The question of how to decide on the parameter t depends on the soundness error the protocol is ready to accept. We discuss this in more detail following the construction.

Notice however that C_i does not want to reveal any information about $\boldsymbol{x}$ to KS. A hiding (or randomized) VC scheme tackles this issue but a hiding vector commitment cannot be used directly as the PRF input. This is because, for the same input $\boldsymbol{x}$, if the PRF is computed on two randomized VC's for $\boldsymbol{x}$, then it will generate different outputs. So we require that PRF is computed on a deterministic vector commitment to the input vector. Let s' be a randomized vector commitment to C_i's input $\boldsymbol{x}$.

We solve problem above by having C_i send both com and s' and a proof that ensures that com and s' are appropriately related, namely that they both refer to the same deterministic vector commitment s.

Armed with this intuition, we are ready to give full construction of the protocol. In Appendix E we give the full implementations of the PoK's used here.

Setup: On input of $(\mathsf{Setup}, \mathsf{sid})$, the key server KS executes:

1. Receive (par) on the $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{PRF.Setup},\mathsf{VC.Setup},\mathsf{CS.Setup},\mathsf{PK.Setup},\mathsf{HES.Setup},\mathsf{sp}}$ interface.[2]
2. Run $k \leftarrow \mathsf{PRF.KeyGen}(1^\lambda, \mathsf{par})$ and store k.
3. Output $(\mathsf{Setup}, \mathsf{sid})$

Evaluate: On input $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathsf{par}.\mathcal{M}^n)$ to client C_i, the following protocol is executed between C_i and KS:

1. Receive (par) on the $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{PRF.Setup},\mathsf{VC.Setup},\mathsf{CS.Setup},\mathsf{PK.Setup},\mathsf{HES.Setup},\mathsf{sp}}$ interface.
2. C_i picks a random $r_1 \leftarrow \mathsf{par}.\mathcal{R}$ and computes $s \leftarrow \mathsf{VC.Commit}(\mathsf{par}, \boldsymbol{x})$ and $s' \leftarrow \mathsf{VC.RandCommitment}(\mathsf{par}, s, r_1)$.
   Additionally, C_i does the following:
   (a) Pick a random $r_2 \leftarrow \mathsf{par}.\mathcal{R}$

(b) Compute $\mathsf{com} \leftarrow \mathsf{CS.Commit}(\mathsf{par}, s, r_2)$

(c) Then, C_i generates the following proof of knowledge

$$\pi_{c00} := \mathsf{PoK}\{(\underline{s}, \underline{r_2}) : \mathsf{com} = \mathsf{CS.Commit}(\mathsf{par}, s, r_2)\}$$

(d) Additionally, C_i computes the following proof of knowledge

$$\pi_{c01} := \mathsf{PoK}\left\{\begin{array}{r}(s, r_1, r_2) : \mathsf{com} = \mathsf{CS.Commit}(\mathsf{par}, s, r_2) \quad \wedge \\ s' = \mathsf{VC.RandCommitment}(\mathsf{par}, s, r_1)\end{array}\right\}$$

and sends $(s', \mathsf{com}, \pi_{c00}, \pi_{c01})$ to KS.

3. KS verifies π_{c00}, π_{c01} if the verifications pass through, then it proceeds to the next step.
4. KS picks a set of indices $I = \{j_1, \ldots, j_t\}$ from $[1, n]$ randomly and sends them to C_i
5. For each challenged index $j \in I$, C_i computes

$$(w'_j, x_j) \leftarrow \mathsf{VC.Prove}(\mathsf{par}, j, \boldsymbol{x}) \ ,$$

$$(w_j) \leftarrow \mathsf{VC.RandWitness}(\mathsf{par}, s, j, r_1, w'_j)$$

and generates the following proof of knowledge

$$\pi_j = \mathsf{PoK}\left\{ (\underline{w_j}, \underline{x_j}) : 1 = \mathsf{VC.Verify}(\mathsf{par}, j, s', w_j, x_j)\right\}$$

Let $\pi_{c1} = \{\pi_j | j \in \mathcal{I}\}$. C_i sends π_{c1} back to KS.

6. KS verifies π_{c1} and if the verification passes, KS proceeds to the next step.
7. KS picks $(\mathsf{epk}, \mathsf{esk}) \leftarrow \mathsf{HES.KeyGen}(\mathsf{par})$, computes $[k] \leftarrow \mathsf{HES.Enc}(\mathsf{epk}, k)$ and sends $\mathsf{epk}, [k]$ to C_i.
8. Then C_i picks $r_3 \in \mathsf{par}.\mathcal{R}$ and computes

$$\mathsf{ct} \leftarrow ([k][s])^{r_3} \ ,$$

where $[s] \leftarrow \mathsf{HES.Enc}(\mathsf{epk}, s)$

9. Next, C_i generates the following proof of knowledge

$$\pi_{c2} = \mathsf{PoK}\left\{\begin{array}{c}(s, r_2, \underline{r_3}) : \mathsf{com} = \mathsf{CS.Commit}(\mathsf{par}, s, r_2) \quad \wedge \\ \mathsf{ct} = ([k][s])^{r_3}\end{array}\right\}$$

C_i sends ct, π_{c2} to KS.

10. KS verifies π_{c2} and if the verifications succeed, KS continues.
11. KS computes $V \leftarrow \mathsf{HES.Dec}(\mathsf{esk}, \mathsf{ct})$
12. Then KS computes $K' \leftarrow \mathsf{PRF.Evaluate}(\mathsf{par}, k, V)$ and sends it to C_i.
13. If $K' = \perp$, output $\perp$. Otherwise compute $K \leftarrow K'^{(r_3 \mod \mathsf{par}.N)}$ and output K.[8]

Choosing parameter t: Parameter t is a tuning parameter that trades communication bandwidth for efficiency. In order to achieve high confidence that the prover (i.e., the client) owns the entire file, t has to be adjusted accordingly. Intuitively, for higher confidence that the prover possesses the entire file, the verifier can set t to a large value. To minimize soundness error, a file can be erasure coded first and then a VC commitment can be computed on the erasure coded file. If the erasure code is resilient to erasures of up to α fraction of the bits and ϵ is the desired soundness bound, then t should be picked as follows: t should be the smallest integer such that $(1-\alpha)^t < \epsilon$. [19] discusses in detail how to tune t. Even though this scheme achieves a high level of soundness, good erasure codes for very large files are expensive to compute. In [19], the authors propose a pairwise hash function with public parameters that can be used to hash the input file down to a constant size and then run VC on it. This scheme achieves a weaker level of security than the erasure coded version.

Discussion on tolerating malicious KS. The functionality is independent of whether or not KS is honest-but-curious or not. This only matters for the implementation and to what extent it realizes the functionality, i.e., our protocol Π_{OOPRF} realizes the functionality under the assumption that KS is honest or honest-but-curious. Notice that, in the functionality, the key server KS does not learn any information about C_i's input by design. So the functionality protects the privacy of C_i's input even from a malicious KS.

The choice of making KS honest-but-curious merits further discussion. In Π_{OOPRF}, KS can be made to commit to its PRF key and to return a proof of knowledge that it has computed the OPRF correctly as Jarecki and Liu do [21]. However, this does not guarantee that KS will pick a strong key or keep that key secret both of which would defeat the purpose of the protocol. Thus, to address a fully malicious KS, we need to ensure that KS has chosen its PRF key by sampling randomly the desired key space. We notice that this is not addressed by Jarecki et al. [21] either, even though they claim to handle fully malicious KS (i.e., PRF evaluator). Handling this aspect is left as future work.

5 Merkle Tree-based Vector Commitment

In this section we present a new VC construction scheme based on the Merkle Hash Tree (MHT) based accumulator construction presented in [4]. Unlike [4], we do not need to hide the index position of the leaf. This allows for some efficiency enhancement since the prover does not need to hide if a node is the left child or the right child of its parent. We first provide a detVC construction

[2] Note that $\mathsf{par} = (\mathsf{par}_{\mathsf{PRF}}, \mathsf{par}_{\mathsf{VC}}, \mathsf{par}_{\mathsf{CS}}, \mathsf{par}_{\mathsf{PK}}, \mathsf{par}_{\mathsf{HES}})$, but by the choice of our schemes, they all work in the same setting with shared parameters. To simplify notation, when the primitive used is clear from the context, we will just refer to par and not to the specific parameters of that primitive.

[8] It is clear that the randomness r_3 cancels out only with algebraic PRF's with appropriate codomains as the one chosen in our construction.

and then describe algorithms $\mathsf{RandCommitment}, \mathsf{RandWitness}$ to convert it to randVC. Notice that in this VC construction, the public parameter is constant-sized as opposed to the CDH and RSA based VC schemes proposed in [13]. The drawback is that the proofs have length logarithmic in n as opposed to constant.

5.1 detVC and randVC Constructions

$\underline{\mathsf{VC.Setup}(1^\lambda, n)}$:On input security parameter 1^λ and an upper bound n, the algorithm invokes $\mathsf{CS.Setup}(1^\lambda)$. Let $\mathsf{CS.Setup}(1^\lambda)$ return $(\rho, N, G, \mathsf{G}, \mathsf{H}, \mathcal{M}, \mathcal{R})$. This algorithm appends the tuple with the collision-resistant hash function $H : (\mathbb{Z}_N)^2 \to \mathbb{Z}_N$ defined as follows [4]: $H(x, y) = x^7 + 3y^7 \mod N$ and return it as par. For further details on the hash function, see [4].

$\underline{\mathsf{VC.Commit}(\mathsf{par}, \boldsymbol{x})}$: On input public parameters par and input $\boldsymbol{x} = x_1, \ldots, x_n$, the algorithm, using $H(\cdot, \cdot)$, recursively builds a Merkle Hash Tree on $\boldsymbol{x}$ (as described in Section A). (If n is not a power of two, insert "dummy" elements into $\boldsymbol{x}$ until n is a perfect power of 2.) Let MR be the root of the MHT. The algorithm outputs commitment $\mathsf{com} = \mathsf{MR}$.

$\underline{\mathsf{VC.Prove}(\mathsf{par}, i, \boldsymbol{x})}$: On input public parameters par, position i and input $\boldsymbol{x} = x_1, \ldots, x_n$, the algorithm does the following: Let us denote the node values along the path from the root node with value MR, to the leaf node, with value $x[i]$, in the MHT as: $\mathcal{P} = (p_0, p_1, \ldots, p_d)$. Note that $p_0 = \mathsf{MR}$ and $p_d = x[i]$. Let $\mathcal{P}_{\mathcal{S}} = (p'_1, \ldots, p'_d)$ be the sibling path of $\mathcal{P}$ (note that p_0 has no sibling). Then, the algorithm computes $\mathcal{P}_{\mathcal{S}}$ and outputs witness $(w = \mathcal{P}_{\mathcal{S}}, x_i)$.

$\underline{\mathsf{VC.Verify}(\mathsf{par}, i, \mathsf{com}, w, x)}$: On input public parameters par, position i, commitment $\mathsf{com} = \mathsf{MR}$, witness (w, x), the algorithm parses w as $\mathcal{P}_{\mathcal{S}} = (p'_1, \ldots, p'_d)$ and sets $p_d = x$. For each $j = d, \ldots 1$, the algorithm recursively computes the internal nodes by hashing the left and right child. Let $p_0 = H(p_1, p'_1)$ (if p_1 is the left sibling, $H(p'_1, p_1)$ otherwise.). This algorithm checks if $\mathsf{MR} = p_0$. It outputs 1 if the equality holds, 0 otherwise.

$\underline{\mathsf{VC.RandCommitment}(\mathsf{par}, \mathsf{com}, r)}$: On input public parameters par, non-hiding vector commitment $\mathsf{com} = \mathsf{MR}$ and randomness $r \in \mathcal{R}$, the algorithm invokes $\mathsf{CS.Commit}(\mathsf{par}, \mathsf{MR}, r)$. Let $\mathsf{CS.Commit}(\mathsf{par}, \mathsf{MR}, r)$ return $(\mathsf{com}_{\mathsf{MR}}, \mathsf{open}_{\mathsf{MR}})$. Output $\mathsf{com}' = \mathsf{com}_{\mathsf{MR}}$.

$\underline{\mathsf{VC.RandWitness}(\mathsf{par}, \mathsf{com}, i, r, w)}$: On input public parameters par, non-hiding vector commitment $\mathsf{com} = \mathsf{MR}$, position i, randomness $r \in \mathcal{R}$ and a deterministic witness w, the algorithm does the following: 1) parses w as $\mathcal{P}_{\mathcal{S}} = ((p'_1, \ldots, p'_d), v)$ 2) computes $(\mathsf{com}_{\mathsf{MR}}, \mathsf{open}_{\mathsf{MR}}) = \mathsf{CS.Commit}(\mathsf{par}, \mathsf{MR}, r)$ 3) computes $w' = (\mathcal{P}_{\mathcal{S}}, v, \mathsf{com}_{\mathsf{MR}}, \mathsf{open}_{\mathsf{MR}})$ and outputs w'.

$\mathsf{VC.Verify}$ *for randomized witness: the only changes in the verification algorithm are the following: 1) parse w as $(\mathcal{P}_{\mathcal{S}} = (p'_1, \ldots, p'_d), x_i, \mathsf{com}_{\mathsf{MR}}, \mathsf{open}_{\mathsf{MR}})$ 2) in the last step instead of checking if* $\mathsf{MR} = p_0$*, check if* $\mathsf{CS.Verify}(\mathsf{par}, \mathsf{com}_{\mathsf{MR}}, \mathsf{MR}, \mathsf{open}_{\mathsf{MR}}) = 1$*. The algorithm will output 1 if the equality holds, 0 otherwise.*

References

1. Algesheimer, J., Camenisch, J., Shoup, V.: Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 417–432. Springer, Heidelberg (Aug 2002)
2. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (May 2004)
3. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to iops and stateless blockchains. Cryptology ePrint Archive, Report 2018/1188 (2018)
4. Boneh, D., Corrigan-Gibbs, H.: Bivariate polynomials modulo composites and their applications. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 42–62. Springer, Heidelberg (Dec 2014)
5. Bootle, J., Cerulli, A., Chaidos, P., Groth, J.: Efficient Zero-Knowledge Proof Systems, pp. 1–31. Springer International Publishing, Cham (2016)
6. Camenisch, J., Dubovitskaya, M., Rial, A.: UC commitments for modular protocol design and applications to revocation and attribute tokens. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 208–239. Springer, Heidelberg (Aug 2016)
7. Camenisch, J., Kiayias, A., Yung, M.: On the portability of generalized Schnorr proofs. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 425–442. Springer, Heidelberg (Apr 2009)
8. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (Aug 2003)
9. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (Aug 1997)
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001)
11. Canetti, R.: Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239 (2003), http://eprint.iacr.org/2003/239
12. Canetti, R.: Universally composable signature, certification, and authentication. In: CSFW. pp. 219–. CSFW '04, IEEE Computer Society, Washington, DC, USA (2004), https://doi.org/10.1109/CSFW.2004.24
13. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 55–72. Springer, Heidelberg (Feb / Mar 2013)
14. Chase, M., Meiklejohn, S.: Déjà Q: Using dual systems to revisit q-type assumptions. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 622–639. Springer, Heidelberg (May 2014)
15. Damgård, I.: Efficient concurrent zero-knowledge in the auxiliary string model. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 418–430. Springer, Heidelberg (May 2000)
16. Dodis, Y., Shoup, V., Walfish, S.: Efficient constructions of composable commitments and zero-knowledge proofs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 515–535. Springer, Heidelberg (Aug 2008)

17. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (Jan 2005)
18. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (Feb 2005)
19. Halevi, S., Harnik, D., Pinkas, B., Shulman-Peleg, A.: Proofs of ownership in remote storage systems. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 11. pp. 491–500. ACM Press (Oct 2011)
20. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In: (EuroS&P). pp. 276–291 (March 2016)
21. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (Mar 2009)
22. Keelveedhi, S., Bellare, M., Ristenpart, T.: Dupless: Server-aided encryption for deduplicated storage. In: USENIX Security 13. pp. 179–194. USENIX (2013), https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/bellare
23. Liu, J., Asokan, N., Pinkas, B.: Secure deduplication of encrypted data without additional independent servers. In: Ray, I., Li, N., Kruegel:, C. (eds.) ACM CCS 15. pp. 874–885. ACM Press (Oct 2015)
24. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (Aug 1990)
25. Meyer, D.T., Bolosky, W.J.: A study of practical deduplication. Trans. Storage 7(4), 14:1–14:20 (Feb 2012), http://doi.acm.org/10.1145/2078861.2078864
26. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (Aug 1992)
27. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (Dec 2008)
28. Visconti, I.: Efficient zero knowledge on the Internet. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006, Part II. LNCS, vol. 4052, pp. 22–33. Springer, Heidelberg (Jul 2006)

A Preliminaries

Collision Resistant Hash Function (CRHF). A family of functions $\mathcal{H}$ is collision resistant if no efficient algorithm can find, on input a random $H \in \mathcal{H}$, two different inputs $x \neq y$ such that $H(x) = H(y)$ (except with probability negligible in the security parameter). We will use the CRHF proposed in [4].

Merkle Hast Tree (MHT). A Merkle hash tree (MHT) [24] provides a succinct *commitment* to a vector, such that it is later possible to *open* and verify individual values in the vector without opening the entire vector. Given a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, a MHT is constructed on it as follows: group the value pairs and then use a CRHF to hash each pair. The hash values are again grouped in pairs and each pair is further hashed, and this process is repeated until only a single hash value remains. This results in a binary tree with the leaves corresponding to the blocks of the vector and the root corresponding to the last remaining hash value. The root serves as the commitment to $\boldsymbol{x}$ and later individual positions can be opened such that the opening can be verified against the root.

A.1 Universally Composable (UC) Security

A protocols that securely realizes a given task f are defined in three steps, as follows.

1. A protocol Π for executing the given task f is formalized. This is called the *real world protocol*.
2. An *ideal functionality*, $\mathcal{F}$ for carrying out f is formalized. In the ideal process the parties do not communicate with each other. Instead they hand their input to $\mathcal{F}$ which computes f on the received inputs and gives back to each player the appropriate output. $\mathcal{F}$ is essentially an incorruptible *trusted party* that is programmed to capture the functionality of f. In this idealized setting, security is inherently guaranteed as any adversary, controlling some of the parties, can only learn/modify the data of corrupted parties.
3. The designed protocol Π is said to securely realize the ideal functionality if the process of running the real world protocol amounts to emulating the ideal process for $\mathcal{F}$.

The last step is formalized by considering an environment $\mathcal{Z}$ that is allowed to provide inputs to all the participants. $\mathcal{Z}$ chooses the inputs of the parties and collects their outputs. In the real world, $\mathcal{Z}$ can communicate freely with an adversary $\mathcal{A}$, controlling the corrupt parties and the communication among them. In the ideal world, $\mathcal{Z}$ interacts with dummy parties who simply relays inputs and outputs between $\mathcal{Z}$ and $\mathcal{F}$ and an ideal adversary Sim also interacting with $\mathcal{F}$. $\mathcal{Z}$ aims to distinguish the case where it receives the outputs produced from a real execution of the protocol from the case where it receives outputs obtained from an ideal execution of the protocol. Π realizes the functionality $\mathcal{F}$ if for every (polynomially bounded) $\mathcal{A}$, there exists a (polynomially bounded) Sim such that no (polynomially bounded) $\mathcal{Z}$ can distinguish a real execution of the protocol from an ideal one with a significant advantage. The universal composability theorem assures that Π continues to behave like the ideal functionality even if it is composed with other arbitrary protocols.

More formally, let $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, a)$ denote the distribution given by the output of $\mathcal{Z}$ on input a with $\mathcal{A}$ and parties running protool Π and $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(\lambda, a)$ denote the distribution given by the output of $\mathcal{Z}$ on input a with Sim and dummy parties relaying to $\mathcal{F}$. Protocol Π safely realizes $\mathcal{F}$ if, for all polynomial-time $\mathcal{A}$ there exists a polynomial-time Sim such that, for all polynomial-time $\mathcal{Z}$, the two distributions $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$ are indistinguishable.

A protocol $\Pi^{\mathcal{G}}$ securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model when Π is allowed to invoke ideal functionality $\mathcal{G}$.

Session ID: In the UC framework there may be many copies of the ideal functionality running in parallel. Each copy is supposed to have a unique session identifier (sid) that describes which session or instance it belongs to. Every time a message has to be sent to a specific copy of $\mathcal{F}$, such a message should contain the sid of the copy it is intended for.

Ideal Functionalitiy $\mathcal{F}_{\mathsf{CRS}}$ [10]: We extend the functionality in [10] by making the common reference string depend on the system parameters. $\mathcal{F}_{\mathsf{CRS}}$ is parameterized by a ppt algorithm $\mathsf{CRS.Setup}$ and by system parameters sp.

Functionality $\mathcal{F}_{\mathsf{crs}}$ (parametarized with system parameters sp): On input $(\mathsf{startCRSgen}, \mathsf{sid})$ from any party P,

- If $(\mathsf{sid}, \mathsf{crs})$ is stored, set $\mathsf{crs}' \leftarrow \mathsf{crs}$. Otherwise, run $\mathsf{crs}' \leftarrow \mathsf{CRS.Setup}(\mathsf{sp})$ and store $(\mathsf{sid}, \mathsf{crs}')$
- Send $(\mathsf{endCRSgen}, \mathsf{sid}, \mathsf{crs}')$ to P.

B $\mathit{\Pi}_{\mathsf{OOPRF}}$ securely realizes $\mathcal{F}_{\mathsf{OOPRF}}$

To prove that our construction $\mathit{\Pi}_{\mathsf{OOPRF}}$ securely realizes the ideal functionality $\mathcal{F}_{\mathsf{OOPRF}}$, we have to show that, for any environment $\mathcal{Z}$ and any adversary $\mathcal{A}$, a simulator Sim exists such that $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and the protocol in the real world or with Sim and $\mathcal{F}_{\mathsf{OOPRF}}$. The simulator plays the role of all honest parties in the real world and interacts with $\mathcal{F}_{\mathsf{OOPRF}}$ for all corrupt parties in the ideal world. In the next section, we present the simulator Sim and the proof of security.

B.1 Simulator.

We are proving the security of $\mathit{\Pi}_{\mathsf{OOPRF}}$ in the $(\mathcal{F}_{1-\mathsf{auth}}, \mathcal{F}_{\mathsf{CRS}})$-hybrid model where we use functionality $\mathcal{F}_{1-\mathsf{auth}}$ (one-side authenticated communication as we want the clients to remain anonymous to KS) [12] to realize the communication between the parties. In this model, Sim will always simulate $\mathcal{F}_{1-\mathsf{auth}}, \mathcal{F}_{\mathsf{CRS}}$ towards the malicious party. Sim simulates $\mathcal{F}_{\mathsf{CRS}}$ as follows:

On input $(\mathsf{startCRSgen}, \mathsf{sid})$ from $\mathsf{Sim}_{\mathsf{CRS}}$, if par is not stored, Sim sets parameters par and trapdoor td as follows.

- Run $(\mathsf{par}_{\mathsf{PRF}}) \leftarrow \mathsf{PRF.Setup}(\mathsf{sp})$
- Run $(\mathsf{par}_{\mathsf{VC}}) \leftarrow \mathsf{VC.Setup}(\mathsf{sp}, n)$
- Run $(\mathsf{par}_{\mathsf{HES}}) \leftarrow \mathsf{HES.Setup}(\mathsf{sp})$
- Run $(\mathsf{par}_{\mathsf{CS}}) \leftarrow \mathsf{CS.Setup}(\mathsf{sp})$
- Run $(\mathsf{par}_{\mathsf{PK}}, \mathsf{td}_s \mathsf{td}_e) \leftarrow \mathcal{SE}(\mathsf{sp})$.
- Set $\mathsf{par} := (\mathsf{par}_{\mathsf{PRF}}, \mathsf{par}_{\mathsf{VC}}, \mathsf{par}_{\mathsf{HES}}, \mathsf{par}_{\mathsf{CS}}, \mathsf{par}_{\mathsf{PK}})$ and $\mathsf{td} := (\mathsf{td}_s, \mathsf{td}_e)$

as we want the client to remain anonymous from KS. a functionality $F_c h$ to realize the communication between party.

We address first the case of a malicious client and honest server, and then the case of an honest client and honest-but-curious server.

Malicious C_i and honest KS: On input $(\mathsf{Setup}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{OOPRF}}$, Sim simulates $\mathcal{F}_{\mathsf{CRS}}$ towards C_i. Then Sim sends $(\mathsf{Setup}, \mathsf{sid}, \mathsf{ok})$ to $\mathcal{F}_{\mathsf{OOPRF}}$.

At some point after the setup, Sim receives $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid})$ from an anonymous client. Here, Sim plays the role of an honest KS with the client as per the protocol. When C_i sends $(s', \mathsf{com}, \pi_{c00}, \pi_{c01})$, then Sim uses the extractor of π_{c00} to extract s (recall that $\pi_{c00} := \mathsf{PoK}\{(\underline{s}, \underline{r_2}) : \mathsf{com} = \mathsf{CS.Commit}(\mathsf{par}, s, r_2)\}$). Then, Sim receives $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{startEvaluate})$ from $\mathcal{F}_{\mathsf{OOPRF}}$ and does the following:

- Picks a set of random t indices: $\mathcal{I} = [i_1, \ldots, i_t]$ from $[1, n]$, where n is the upper bound on the length of the files send them to C_i.
- Let C_i respond with $\pi_{c1} = \{\pi_j | j \in \mathcal{I}\}$. Sim extracts (x_j, w_j) from each π_j and
- Runs $b \leftarrow \mathsf{detVC.Verify}(s, x[j], w_j)$ for all $j \in \mathcal{I}$. If at least one verification fails, Sim aborts.
- Else, sends $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{startEvaluateok}, s, \{x_j, w_j\}_{j \in \mathcal{I}})$ to $\mathcal{F}_{\mathsf{OOPRF}}$

Finally, Sim gets back $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, r)$ from $\mathcal{F}_{\mathsf{OOPRF}}$.Now, if the verification of π_{c2} fails then Sim returns $\perp$. Otherwise, Sim sets $K' \leftarrow r$ and forwards it to C_i.

Honest C_i and honest-but-curious KS: $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{PRF.Setup}, \mathsf{VC.Setup}, \mathsf{CS.Setup}, \mathsf{PK.Setup}, \mathsf{HES.Setup}, \mathsf{sp}}$ is simulated by Sim towards KS. Sim will then invoke $\mathcal{F}_{\mathsf{OOPRF}}$ with $(\mathsf{Setup}, \mathsf{sid})$. Sim recieves $(\mathsf{Setup}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{OOPRF}}$ and responds with $(\mathsf{Setup}, \mathsf{sid}, \mathsf{ok})$.

Then, Sim runs the **Evaluate** part of Π_{OOPRF} playing the role of an honest client C_i. Sim does that by picking a random file $\tilde{\boldsymbol{x}}$, simulates the proofs for the challenged positions and runs the necessary steps of the protocol.

At some point, Sim receives $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{startEvaluate})$ from $\mathcal{F}_{\mathsf{OOPRF}}$ and if the protocol execution above succeeded, then Sim sends $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{startEvaluate}, \mathsf{ok})$ to $\mathcal{F}_{\mathsf{OOPRF}}$ causing $\mathcal{F}_{\mathsf{OOPRF}}$ to output to C_i.

B.2 Proof of Security

Theorem 4. *In the $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{PRF.Setup}, \mathsf{VC.Setup}, \mathsf{CS.Setup}, \mathsf{PK.Setup}, \mathsf{HES.Setup}, \mathsf{sp}}$-hybrid model, protocol Π_{OOPRFPSF} securely realizes $\mathcal{F}_{\mathsf{OOPRF}}$ if the underlying vector commitment scheme VC is hiding and binding, CS is a hiding and binding commitment scheme, HES is a semantically secure encryption scheme and the non-interactive proof of knowledge scheme PK is zero knowledge and simulation-sound extractable.*

Proof. We now show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the ensemble $\mathsf{REAL}_{\Pi_{\mathsf{OOPRF}}, \mathcal{A}, \mathcal{Z}}$ and the ensemble $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{OOPRF}}, \mathsf{Sim}, \mathcal{Z}}$ with non-negligible probability. The idea is that we start with the real world protocol and then modify it until the simulator incorporates $\mathcal{F}_{\mathsf{OOPRF}}$.

We first address the case of malicious client and honest server and then the case of a honest client and honest-but-curious server. At a high level, the proof for the malicious client and honest server case requires simulation sound extractability of the underlying proof system, IND-CPA security of the homomorphic encryption scheme and the pseudorandomness of the PRF. Leveraging each of these security properties at the intermediate game hops, we move from the ensemble $\mathsf{REAL}_{\Pi_{\mathsf{OOPRF}},\mathcal{A},\mathcal{Z}}$ to the ensemble $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{OOPRF}},\mathsf{Sim},\mathcal{Z}}$.

The honest client and honest-but-curious server is simpler. It requires simulation soundness of the proof system to be able to simulate the proofs for the challenged positions and hiding property of the randomized vector commitment scheme. Leveraging these security properties at the intermediate game hops, we move from the ensemble $\mathsf{REAL}_{\Pi_{\mathsf{OOPRF}},\mathcal{A},\mathcal{Z}}$ to the ensemble $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{OOPRF}},\mathsf{Sim},\mathcal{Z}}$.

Malicious C_i and honest KS:

Game 0: This game corresponds to the execution of the real-world protocol Π_{OOPRF}.

Game 1: Same as the previous game, except the following: Game 1 replaces the parameters $\mathsf{par}_{\mathsf{PK}}$ computed by $\mathsf{PK.Setup}$ by parameters computed by $\mathcal{SE}$. By the simulation-sound extractability of the PoK, Game 0 is indistinguishable from Game 1.

Game 2: Same as the previous game, except the following: Game 2 calls the extractor of the PoK π_{c00} to extract s. The probability that the extractor extracts s is non-negligible by the simulation-sound extractability of the PoK. Therefore, Game 1 is indistinguishable from Game 2.

Game 3: Same as the previous game, except the following: Game 3 uses the extractor of π_j for all $\pi_j \in \pi_{c1}$[9] to extract (x_j, w_j). By the simulation-sound extractability of the PoK, the extractor will extract (x_j, w_j) with non-negligible probability.

Now notice that if for some $j \in \mathcal{I}$, the malicious client is able to produce, in two different runs, $(x_j, w_j), (x'_j, w'_j)$ such that

$$x_j \neq x'_j, w_j \neq w'_j \ ,$$

and still

$$\mathsf{VC.Verify}(\mathsf{par}, s', j, x_j, w_j) = \mathsf{VC.Verify}(\mathsf{par}, s', j, x'_j, w'_j) = 1 \ ,$$

then the real protocol Π_{OOPRF} will accept and move to the next step whereas the ideal functionality $\mathcal{F}_{\mathsf{OOPRF}}$ will reject and stop.
Therefore, $\mathcal{Z}$'s views will clearly be different in this two cases and $\mathcal{Z}$ can use this strategy to distinguish between the real and the ideal world.

We prove that this can happen only with negligible probability by reducing to the binding property of VC scheme. The reduction can use $(x_j, w_j, x'_j, w'_j, s)$ as a response to the VC binding game, thereby breaking VC's binding. Hence, this happens with negligible probability and $\mathcal{Z}$'s view in Game 2 is indistinguishable from $\mathcal{Z}$'s view in Game 3.

[9] Notice that actually, the tuples (x_j, w_j) are extracted once at the time. Therefore, $t = \mathsf{poly}(\lambda)$ sub games are needed. We did not include them to avoid burdening the flow of the proof.

Game 4: Same as the previous game, except the following: Game 4 calls the extractor of the PoK π_{c2} to extract r_3. By the simulation-sound extractability of the PoK, π_{c2}, the probability that the extractor extracts r_3 is non-negligible. $\mathcal{Z}$'s view in these two games are indistinguishable.

Game 5: Same as the previous game, except the following: Game 5, instead of decrypting ct, computes K' as follows. Notice that if a malicious client sends a malformed ciphertext, then π_{c2} will not verify and the Sim will abort except with negligible probability.

1. If $\mathsf{gcd}(k+s, N) \neq 1$, send $\perp$ to C_i and abort. Else proceed to the next step.
2. Set $K' \leftarrow g^{\frac{1}{(k+s)r_3} \mod N}$

Note that if $\mathsf{gcd}(k+s, N) \neq 1$ then $\beta = r_3(k+s)$ is not co-prime with N for any r_3, so $\mathsf{gcd}(\mathsf{HES.Dec}(\mathsf{epk}, \mathsf{ct}), N) \neq 1$. Hence, in the real protocol, the KS will also send $\perp$. Thus $\mathcal{Z}$'s views in these two games are indistinguishable.

Game 6: Same as the previous game, except the following: In Game 6, $[k]$ is replaced with $[k']$ for some random k' in the PRF key space.

If $\mathcal{Z}$ can distinguish between these two games, then we can use $\mathcal{Z}$ to break the IND-CPA security of the underlying encryption scheme.

In more detail, the reduction receives epk from the IND-CPA challenger and pick a random $k' \neq k$. It then sends k, k' as its challenge messages and receive c as its challenge ciphertext. Sim sends epk and c to C_i as $[k]$. If the challenge ciphertext contains k, then the reduction simulates Game 4, else it simulates Game 5. If $\mathcal{Z}$ can distinguish between these two games, then the reduction uses $\mathcal{Z}$ to break the IND-CPA security.
Hence $\mathcal{Z}$'s views in these two games are indistinguishable.

Game 7: Same as the previous game, except the following: In Game 7 the computation of K' is replaced by the following computation: $K' \leftarrow g^r$ for some random $r \in \mathcal{R}$. If $\mathcal{Z}$ can distinguish the between these two games, then it breaks the security of the PRF.

In details, the reduction gets oracle access to either the PRF or a random function. To compute K', the reduction invokes its oracle on input s and receives output v. Then the reduction sets $K' \leftarrow v^{(r_3{}^{-1} \mod N)}$ (r_3 was extracted in Game 4). If the oracle was the PRF, then the reduction is simulating Game 6, else Game 7. So, if $\mathcal{Z}$ can distinguish between this two games, then the reduction uses this $\mathcal{Z}$ to break the PRF security.
Hence, $\mathcal{Z}$'s views in this two games are indistinguishable.

Game 8: Same as the previous game, except the following: instead of using K', Game 8, uses the key it receives from $\mathcal{F}_{\mathsf{OOPRF}}$ as K'. Clearly, $\mathcal{Z}$'s views in these two games are indistinguishable.
The distribution produced by Game 8 is identical to that of our simulation. Sim interacts with the adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\mathsf{OOPRF}}$ as described above.

Honest C_i and honest-but-curious KS:

Game 0: This game corresponds to the execution of the real-world protocol Π_{OOPRF}.

Game 1: Same as the previous game, except the following: Game 1 replaces the parameters $\mathsf{par}_{\mathsf{PK}}$ computed by $\mathsf{PK.Setup}$ by parameters computed by $\mathcal{SE}$. By the simulation-sound extractability of the PoK, Game 0 is indistinguishable from Game 1.

Game 2: Same as the previous game, except the following: Game 2 generates the proof for each position by running the simulator for the proof system[10]. The zero-knowledge property ensures that proofs computes by the simulator are indistinguishable from the proofs computed by the prove algorithm of the PoK system. So $\mathcal{Z}$'s views in these two games are indistinguishable.

Game 3: Same as the previous game, except the following: Game 3 picks a a random vector $\tilde{\boldsymbol{x}}$. At this point, the proofs of positions are simulated proofs. The hiding property of the underlying randVC scheme guarantees that a commitment to a given vector is indistinguishable from a commitment to a random vector. Hence $\mathcal{Z}$'s views in these two games are indistinguishable.

The distribution produced by Game 3 is identical to that of our simulation. Sim interacts with the adversary KS and the ideal functionality $\mathcal{F}_{\mathsf{OOPRF}}$ as described above. Sim sends $(\mathsf{Evaluate}, \mathsf{sid}, \mathsf{qid}, \mathsf{endEvaluate}, \mathsf{ok})$ to $\mathcal{F}_{\mathsf{OOPRF}}$ if all the protocol steps succeed causing $\mathcal{F}_{\mathsf{OOPRF}}$ to output to C_i.

□

C Security Proof of the VC in Section 5

Theorem 5. *The* VC *scheme satisfies correctness and binding, if* CS *is hiding and binding and* H *is a CRHF.*

Proof. (Sketch) Correctness is easy to see.

Hiding (for the randVC): The scheme is information theoretically hiding by the hiding property of Pedersen commitment.

Binding: Here we prove that if H is a CRHF, then the scheme above is binding. Given an adversary $\mathcal{A}$ that breaks the binding property with non-negligible probability, we construct an algorithm $\mathcal{A}'$ that breaks the CR with non-negligible probability. $\mathcal{A}$ receives the description of H and picks the rest of the parameters as per the construction and send par to $\mathcal{A}$. $\mathcal{A}$ returns a commitment MR, a position index i and two (x, w, x', w') for position i such that the verification passes but $x \neq x'$ and $w \neq w'$. Let us denote the two witness paths $\mathcal{P}_1 = (p_1, \ldots, p_d)$ and $\mathcal{P}_2 = (p'_1, \ldots, p'_d)$ and p_0, p'_0 as the two recovered roots from $\mathcal{P}_1, \mathcal{P}_2$ respectively. Since $p_0 = p'_0 = \mathsf{MR}$ and $p_d \neq p'_d$, it must be the case that there is at least one node where the paths collide. More formally, there exists $1 \leq j \leq d-1$ such that $p_j = p'_j$ even though their children are not equal, i.e., $L_j \neq L'_j \vee R_j \neq R'_j$. $\mathcal{A}'$ outputs $(L_j, R_j), (L'_j, R'_j)$ as the collision pair.

[10] Again, we avoided here the intermediate games to generate the simulated proofs once at the time just for the sake of clarify.

D RSA-based Vector Commitment

In this section we recall the RSA based non-hiding VC scheme presented in [13] with two main changes: (1) We make the commitment scheme *hiding* and (2) we add a PoK to prove i^{th} index instead of providing the value and the witness directly. This is the second instantiation VC that can be plugged in to Π_{OOPRF}. We give concrete implementations of all the accompanying PoK's in Appendix E.

D.1 Construction

We now recall the RSA-based vector commitment construction in [13] and point out that these vector commitments are randomizable.

$\mathsf{VC.Setup}(1^\lambda, n)$: On input security parameter 1^λ and an upper bound n, the algorithm does the following:

1. Pick two λ bit safe primes $p = 2p' + 1$ and $q = 2q' + 1$ and set $N = pq$.
2. Pick $l = \mathsf{poly}(\lambda)$ to be the upper bound on the length of the messages.
3. Choose n, $l + 161$[11] bit primes $\mathsf{e}_1, \dots, \mathsf{e}_n$ that do not divide $\phi(N)$.
4. Pick a random $\mathsf{b} \leftarrow \mathbb{Z}_N^*$.
5. For $j = 1, \dots, n$, compute $\mathsf{K}_j \leftarrow \mathsf{b}^{\prod_{i=1, i \neq j}^{n} \mathsf{e}_i} \mod N$
6. Set $\mathsf{K}_0 \leftarrow \mathsf{b}^{\prod_{i=1}^{n} \mathsf{e}_i} \mod N$
7. Output the parameters $\mathsf{par} := (N, \mathsf{b}, \mathsf{e}_1, \dots, \mathsf{e}_n, \mathsf{K}_0, \dots, \mathsf{K}_n, \mathcal{M} = \{0,1\}^l, \mathcal{R} = \mathbb{Z}_N^*)$

$\mathsf{VC.Commit}(\mathsf{par}, \boldsymbol{x})$: On input public parameters par and input $\boldsymbol{x} = x_1, \dots, x_n$, the algorithm computes $\mathsf{com} \leftarrow \mathsf{K}_1^{x_i} \dots \mathsf{K}_n^{x_n} \mod N$ and outputs commitment com.

$\mathsf{VC.Prove}(\mathsf{par}, i, \boldsymbol{x})$: On input public parameters par, position i and input $\boldsymbol{x} = x_1, \dots, x_n$, the algorithm computes $w \leftarrow (\prod_{j=1, j \neq i}^{N} \mathsf{K}_j{}^{x[j]})^{\frac{1}{\mathsf{e}_i}} \mod N$ and outputs witness (w, x_i).

$\mathsf{VC.Verify}(\mathsf{par}, i, \mathsf{com}, w, x)$: On input public parameters par, position i, commitment com, witness (w, x), the algorithm outputs 1 if $\mathsf{K}_i^x w^{\mathsf{e}_i} \mod N = \mathsf{com}$ and $x \in \mathcal{M}$, 0 otherwise.
Note that the verification algorithm is the same for both the randVC *and the* detVC *versions. Only instead of* com*, the algorithm will take* com' *as input.*

$\mathsf{VC.RandCommitment}(\mathsf{par}, \mathsf{com}, r)$: On input public parameters par, non-hiding vector commitment com and randomness $r \in \mathcal{R}$, the algorithm computes $\mathsf{com}' \leftarrow \mathsf{com} \cdot \mathsf{K}_0^r \mod N$ and outputs com'.

$\mathsf{VC.RandWitness}(\mathsf{par}, \mathsf{com}, i, r, w)$: On input public parameters par, non-hiding vector commitment com, position i, randomness $r \in \mathcal{R}$ and partial witness w, the algorithm computes $w' \leftarrow w \cdot (\mathsf{b}^{\prod_{j=1, j \neq i}^{n} \mathsf{e}_j})^r = w \cdot \mathsf{K}_0{}^{\frac{r}{\mathsf{e}_i}} \mod N$ and outputs w'.

[11] We require this to be larger than the message length l, But for an efficient range proof that the message is within the correct range, we need to allow the buffer for the length of the challenge space and the statistical parameter, which is 160 bits total. The implementation is in Section E.

D.2 Security Proof

Theorem 6. *The* VC *scheme presented above is correct, hiding and binding under the RSA assumption.*

Proof. (Sketch) The completeness is easy to see.
Hiding (for the randVC): Hiding is satisfied information theoretically since the blinding factor is completely random.
Binding: If the RSA assumption holds, then the scheme defined above is binding. More precisely, if there exists an efficient adversary that produces two valid openings to two different messages at the same position, then we can construct a PPT adversary that breaks the RSA assumption. The reduction is identical to the reduction proof of Theorem 6 in [13]. We note that the sole difference of our construction from that of [13] is that we have the hiding factor K_0^r. But the reduction uses the equality ${\mathsf{K}_i}^x w^{\mathsf{e}_i} = {\mathsf{K}_i}^{x'} w'^{\mathsf{e}_i}$ to extracts response for the RSA challenge; so the effect of the hiding factor cancels out in the reduction.

E GSPK Proofs

In the following, we give the concrete implementations of our PoK protocols. To this end we require the CRS to contain the public key of the CPA version of the Camenisch-Shoup encryption scheme [8]. We already have the modulus N in the CRS which we can use. Recall that N is a product of two safe primes which can be generated distributedly [1]. Furthermore, let g' and y' and be a random elements of $\mathbb{Z}_{N^2}^*$ contained in the CRS and set $\mathsf{g} = \mathsf{g}'^{2N}$, $\mathsf{y} = \mathsf{y}'^{2N}$, and $\mathsf{h} = 1 + N \bmod N^2$. We first describe the implementations of the PoK's that are common to both the VC instantiations we presented and then give the implementations specific to each VC.

E.1 GSPK common to both the VC's:

In this Section we show how proof protocols π_{c00} and π_{c2} are realized. More specifically, the proof protocol

$$\pi_{c00} = \mathsf{PoK}\{(\underline{s}, \underline{r}) : \mathsf{com} = \mathsf{CS.Commit}(\mathsf{par}, s, r)\}$$

is realized by first computing $\mathsf{E}_s = (\mathsf{g}^{r_1} \bmod N^2, \mathsf{h}^s\mathsf{y}^{r_1} \bmod N^2)$, $\mathsf{E}_r = (\mathsf{g}^{r_2} \bmod N^2, \mathsf{h}^r\mathsf{y}^{r_2} \bmod N^2)$, and with r_1 and r_2 being randomly drawn from $[N/4]$, sending these values to the verifier, and the executing the following proof protocol with the verifier:

$$\mathsf{GSPK}\{(s, r, r_1, r_2) : \mathsf{com} = \mathsf{G}^s\mathsf{H}^r \wedge \mathsf{E}_s = (\mathsf{g}^{r_1}, \mathsf{h}^s\mathsf{y}^{r_1}) \wedge \mathsf{E}_r = (\mathsf{g}^{r_2}, \mathsf{h}^r\mathsf{y}^{r_2})\}$$

where we have dropped $\bmod \rho$ and $\bmod N^2$ from the terms for brevity.

On the other hand, the proof protocol

$$\pi_{c2} \;=\; \mathsf{PoK}\{(s, r_2, \underline{r_3}) \;:\; \mathsf{com} \;=\; \mathsf{CS.Commit}(\mathsf{par}, s; r_2) \wedge \mathsf{ct} \;=\; ([k][s])^{r_3}\}$$

is realized as follows: Let us denote $[k] = (\mathsf{e}_1, \mathsf{e}_2)$. The prover first computes $\mathsf{E}_r = (\mathsf{g}^{u_r} \bmod N^2, \mathsf{h}^{r_3}\mathsf{y}^{u_r} \bmod N^2)$, where u_r being randomly drawn from $[N/4]$, sends these values to the verifier, and executes the following proof protocol with the verifier:

$$\begin{aligned}\mathsf{GSPK}\{(s, r_2, r_3, w, r) : \mathsf{com} = \mathsf{G}^s\mathsf{H}^{r_2} \wedge 1 = \mathsf{com}^{-r_3}\mathsf{G}^w\mathsf{H}^{r'} \wedge \\ \mathsf{ct} = (\mathsf{e}_1^{r_3}\mathsf{epk}^B\mathsf{h}^w, \mathsf{e}_2^{r_3}\mathsf{g}^B) \wedge \mathsf{E}_r = (\mathsf{g}^{u_r}, \mathsf{h}^{r_3}\mathsf{y}^{u_r})\}\end{aligned}$$

Here, the term $1 = \mathsf{com}^{-r_3}\mathsf{G}^w\mathsf{H}^{r'}$ shows that $w = sr_3$ and hence that $\mathsf{ct} = ([k][s])^{r_3}$ with B being the value that the prover used to randomize the encryption.

E.2 GSPK for the MHT-VC:

For our OOPRF scheme we need three accompanying PoK's that we need to implement efficiently. Here, we explain which proofs can actually be avoided for the MHT- VC instantiation, and which need more care. Notice that, VC.RandCommitment is the same as CS.Commit algorithm, which computes a Pedersen commitment to the detVC, MR. So, π_{c01} will be just a standard proof of equality [8]. In fact, the following optimisation can be done: use s' as com throughout the protocol and skip π_{01}.

For the proofs π_j: $\mathsf{PoK}\left\{(\underline{w}, \underline{x}) : 1 = \mathsf{randVC.Verify}(\mathsf{par}, j, \mathsf{com}, w, x)\right\}$, proving these relations is a bit more involved and requires the following steps:

1. The algorithm parses w as $(\mathcal{P}_{\mathcal{S}} = (p'_1, \ldots, p'_d), \mathsf{com}_{\mathsf{MR}}, \mathsf{open}_{\mathsf{MR}})$. Let us denote the node values along the path from the root node with value MR, to the leaf node, with value x_i, in the MHT as: $\mathcal{P} = (p_0, p_1, \ldots, p_d)$. The algorithm recovers this path recursively bottom up using $H(\cdot, \cdot)$ on $\mathcal{P}_{\mathcal{S}}$. Note that the index position j uniquely decides the left and the right child at each step.
2. Then, the algorithm commits to every value p_j in this path and to the values of the left and right children of p_j in the MHT, i.e., if l_j is the left child and r_j is the right child of p_j, then the algorithm computes

$$\begin{aligned}(P_j, s_j) \leftarrow \mathsf{CS.Commit}(\mathsf{par}, p_j, s_j), (L_j, s'_j) \leftarrow \mathsf{CS.Commit}(\mathsf{par}, l_j, s'_j), \\ (R_j, s''_j) \leftarrow \mathsf{CS.Commit}(\mathsf{par}, r_j, s''_j)\end{aligned}$$

3. Then, the algorithm generates a proof that P_0 is indeed a commitment to the root.[12]

$$\mathsf{PoK}_{\mathsf{MR}}\{(\underline{\mathsf{MR}}, \underline{r}, \underline{s}) : \mathsf{com} = \mathsf{CS.Commit}(\mathsf{par}, \mathsf{MR}, r) \wedge P_0 = \mathsf{CS.Commit}(\mathsf{par}, \mathsf{MR}, s)\}$$

[12] We are going to abuse the notation a little and ignore the open in the output of CS.Commit for notational convenience.

4. Next, for $j = 0, \ldots, d-1$, the following proof of knowledge that each triplet (P_j, L_j, R_j) is well formed. Note that L_j (or R_j) is used as P_{j+1}.

$$\mathsf{PoK}_j\{(\underline{l}, \underline{r}, s, s', s'') : P_j = \mathsf{CS.Commit}(\mathsf{par}, l^7 + 3r^7, s) \wedge \\ L_j = \mathsf{CS.Commit}(\mathsf{par}, l, s') \wedge R_j = \mathsf{CS.Commit}(\mathsf{par}, r, s'')\}$$

   This proof requires some sub steps which are the following:

   (a) This proof uses the homomorphic property of Pedersen commitment scheme and a subprotocol for $\mathsf{PoK}_{\mathsf{mult}}$ for multiplication of two values. This protocol is instantiated using standard techniques [8, 5].

$$\mathsf{PoK}_{\mathsf{mult}}\{(\underline{x}, \underline{y}, \underline{z}, s_x, s_y, s_z) : C_x = \mathsf{CS.Commit}(\mathsf{par}, x, s_x) \wedge \\ C_y = \mathsf{CS.Commit}(\mathsf{par}, y, s_y) \wedge C_z = \mathsf{CS.Commit}(\mathsf{par}, z, s_z) \wedge z = x \cdot y\}$$

   (b) The prover computes $C_l, C_{l^2}, C_{l^4}, C_{l^6}, C_{l^7}$ and $C_r, C_{r^2}, C_{r^4}, C_{r^6}, C_{r^7}$ and invokes $\mathsf{PoK}_{\mathsf{mult}}$ on each of the following triplets to prove the correctness of the commitments and sends them to the verifier.

$$(C_l, C_l, C_{l^2}) \quad (C_{l^2}, C_{l^2}, C_{l^4}) \quad (C_{l^2}, C_{l^4}, C_{l^6}) \quad (C_l, C_{l^6}, C_{l^7}) \\ (C_r, C_r, C_{r^2}) \quad (C_{r^2}, C_{r^2}, C_{r^4}) \quad (C_r, C_{r^4}, C_{r^6}) \quad (C_r, C_{r^6}, C_{r^7})$$

Now we show how to realize proof protocols $\mathsf{PoK}_{\mathsf{MR}}$ and $\mathsf{PoK}_{\mathsf{mult}}$ for our MHT-VC. In details, the proof protocol

$$\mathsf{PoK}_{\mathsf{MR}}\{(\underline{\mathsf{MR}}, \underline{r}, \underline{s}) : \mathsf{com} = \mathsf{CS.Commit}(\mathsf{par}, \mathsf{MR}, r) \wedge \\ P_0 = \mathsf{CS.Commit}(\mathsf{par}, \mathsf{MR}, s)\}$$

is done as first computing $\mathsf{E}_{\mathsf{MR}} = (\mathsf{g}^{r_1} \bmod N^2, \mathsf{h}^{\mathsf{MR}}\mathsf{y}^{r_1} \bmod N^2)$, $\mathsf{E}_r = (\mathsf{g}^{r_2} \bmod N^2, \mathsf{h}^r\mathsf{y}^{r_2} \bmod N^2)$, and $\mathsf{E}_s = (\mathsf{g}^{r_3} \bmod N^2, \mathsf{h}^s\mathsf{y}^{r_3} \bmod N^2)$, with r_1, r_2, and r_3 being randomly drawn from $[N/4]$, sending these values to the verifier, and then executing the following proof protocol with the verifier

$$\mathsf{GSPK}\{(\mathsf{MR}, r, s, r_1, r_2, r_3) : \mathsf{com} = \mathsf{G}^{\mathsf{MR}}\mathsf{H}^r \ \wedge \ P_0 = \mathsf{G}^{\mathsf{MR}}\mathsf{H}^r \ \wedge \\ \mathsf{E}_{\mathsf{MR}} = (\mathsf{g}^{r_1}, \mathsf{h}^{\mathsf{MR}}\mathsf{y}^{r_1}) \ \wedge \ \mathsf{E}_r = (\mathsf{g}^{r_2}, \mathsf{h}^r\mathsf{y}^{r_2}) \ \wedge \ \mathsf{E}_s = (\mathsf{g}^{r_3}, \mathsf{h}^s\mathsf{y}^{r_3})\} \ ,$$

where we have dropped $\bmod \rho$ and $\bmod N^2$ from the terms for brevity.

On the other hand, proof protocol

$$\mathsf{PoK}_{\mathsf{mult}}\{(\underline{x}, \underline{y}, \underline{z}, s_x, s_y, s_z) : C_x = \mathsf{CS.Commit}(\mathsf{par}, x, s_x) \wedge \\ C_y = \mathsf{CS.Commit}(\mathsf{par}, y, s_y) \wedge C_z = \mathsf{CS.Commit}(\mathsf{par}, c_z, s_z) \wedge z = x \cdot y\}$$

is done by first computing $\mathsf{E}_x = (\mathsf{g}^{u_1} \bmod N^2, \mathsf{h}^x\mathsf{y}^{u_1} \bmod N^2)$ and $\mathsf{E}_y = (\mathsf{g}^{u_2} \bmod N^2, \mathsf{h}^y\mathsf{y}^{u_2} \bmod N^2)$, with r_1 and r_2 being randomly drawn from $[N/4]$, sending these values to the verifier, and then executing the following proof protocol with the verifier:

$$\mathsf{GSPK}\{(x, y, z, s_x, s_y, s_z, s', u_1, u_2) : C_x = \mathsf{G}^x\mathsf{H}^{s_x} \wedge\ C_y = \mathsf{G}^y\mathsf{H}^{s_y}\wedge \\ C_z = \mathsf{G}^z\mathsf{H}^{s_z} \wedge\ C_z = C_y^x\mathsf{H}^{s'} \wedge\ \mathsf{E}_x = (\mathsf{g}^{u_1}, \mathsf{h}^x\mathsf{y}^{u_1}) \wedge\ \mathsf{E}_y = (\mathsf{g}^{u_2}, \mathsf{h}^y\mathsf{y}^{u_2})\}\ .$$

Notice that we do not need to verifiably encrypt the witness z as this one can be computed from x and y. Similarly, a number of encryptions can be dropped when combing these proofs into the bigger proof of the hash-tree path. We leave these optimizations to the reader.

E.3 GSPK for the RSA-VC:

In this section we show how to realize proof protocols π_{c01} and π_j for our RSA-VC. In details, the following statement

$$\pi_{c01} := \mathsf{PoK}\{(s, r_1, r_2) : \mathsf{com}_s = \mathsf{CS.Commit}(\mathsf{par}, s, r_2) \quad \wedge \\ s' = \mathsf{VC.RandCommitment}(\mathsf{par}, s; r_1)\}$$

proves that s' is a randomized version of the vector commitment s to which in turn com_s commits. In other words, that $s' = s\mathsf{K}_0^{r_0}$ holds for some value of r_0 and for the s committed in com_s. This statement can be proved by the following proof protocol

$$\mathsf{GSPK}_{01}\{(r_1, r_2') : \mathsf{G}^{s'} = \mathsf{com}_s^{{\mathsf{K}_0}^{r_1}}\mathsf{H}^{r_2'}) \mod \rho\}\ .$$

Here r_2' absorbs the randomness $-r_2{\mathsf{K}_0}^{r_1}$, i.e., $r_2' = -r_2{\mathsf{K}_0}^{r_1}$, but we do need to prove anything about r_2'. Note that this protocol only works with binary challenges as it involved a "double discrete logarithm" relation (thus the suffice 01 for GSPK_{01}).

On the other hand, for proof

$$\pi_j = \mathsf{PoK}\{(\underline{w}, \underline{x}) : 1 = \mathsf{VC.Verify}(\mathsf{par}, \mathsf{com}', j, x, w)\}$$

we further require that the CRS also contain elements z_1, z_2, and z_3 from $\mathbb{Z}_N^*$ of order $p'q'$ so that that w can be verifiably ElGamal-encrypted w.r.t. to z_1, and to do range proofs for x w.r.t. z_2 and z_3. Thus, the prover first computes $\mathsf{E}_1 = w\mathsf{z}_1^r \bmod N$, $\mathsf{E}_2 = \mathsf{z}_2^r \bmod N$, $\mathsf{E}' = \mathsf{z}_2^x\mathsf{z}_3^r \bmod N$, and $\mathsf{E}_x = (\mathsf{g}^{u_x} \bmod N^2, \mathsf{h}^x\mathsf{y}^{u_x} \bmod N^2)$, where u_x and r being randomly drawn from $[N/4]$, sends E_1, E_2, E', and E_x to the verifier and executes the proof with it

$$\mathsf{GSPK}\{(x, w, r) : \mathsf{com}'/\mathsf{E}_1^{\mathsf{e}_j} = \mathsf{K}_i^x({\mathsf{z}_1}^{-\mathsf{e}_j})^r \wedge \mathsf{E}_2 = \mathsf{z}_2^r\wedge \\ \mathsf{E}_x = (\mathsf{g}^{u_x}, \mathsf{h}^x\mathsf{y}^{u_x}) \wedge \mathsf{E}' = \mathsf{z}_2^x\mathsf{z}_3^r \wedge x \in [-2^{l+160}, 2^{l+160}]\}\ .$$

This proofs shows that $\mathsf{e}_1/\mathsf{z}_1^r$ is a witness for x and that x is in the required range. The witnesses w and x can be on-line extracted by having the simulator set the CRS such that it knows $\log_{\mathsf{z}_1}\mathsf{z}_2$ and $\log_{\mathsf{g}}\mathsf{y}$, respectively.