# Encapsulated Search Index: Public-Key, Sub-linear, Distributed, and Delegatable

Erik Aronesty
Atakama

David Cash
University of Chicago

Yevgeniy Dodis
New York University

Daniel H. Gallancy
Atakama

Christopher Higley
Atakama

Harish Karthikeyan
New York University

Oren Tysor
Atakama

January 18, 2022

**Abstract**

We build the first *sub-linear* (in fact, potentially constant-time) *public-key* searchable encryption system:

- server can publish a public key $PK$.
- anybody can build an encrypted index for document $D$ under $PK$.
- client holding the index can obtain a token $z_w$ from the server to check if a keyword $w$ belongs to $D$.
- search using $z_w$ is almost as fast (e.g., sub-linear) as the non-private search.
- server granting the token does not learn anything about the document $D$, beyond the keyword $w$.
- yet, the token $z_w$ is specific to the pair $(D, w)$: the client does not learn if other keywords $w' \neq w$ belong to $D$, or if $w$ belongs to other, freshly indexed documents $D'$.
- server cannot fool the client by giving a wrong token $z_w$.

We call such a primitive *Encapsulated Search Index* (ESI). Our ESI scheme can be made $(t, n)$-distributed among $n$ servers in the best possible way: *non-interactive*, verifiable, and resilient to any coalition of up to $(t-1)$ malicious servers. We also introduce the notion of *delegatable* ESI and show how to extend our construction to this setting.

Our solution — including public indexing, sub-linear search, delegation, and distributed token generation — is deployed as a commercial application by Atakama.

# Contents

# 1 Introduction

Imagine the user Alice has a powerful but potentially insecure device, which we call *Desktop*. Since the Desktop is insecure (at least when not used by Alice), Alice cannot permanently store any secret keys on the Desktop. Instead, all the secret keys she will need for her work should be stored on a more secure, but weaker device, which we call *Phone*.

Alice works on the Desktop and periodically generates large documents $D_1$, $D_2 \ldots$, that she might want to index separately.[1] Since the documents are sensitive, Alice will always keep the indices encrypted, with the secret key stored on the Phone (and capable of supporting multiple documents $D_1, D_2, \ldots$ with the same key). Moreover, when the Phone approves her search request for keyword $w$ inside the document $D$, the token $z_w$ should only tell if $w \in D$, but will not reveal anything else: either about different keywords $w'$ in $D$, or the same keyword $w$ for another document $D'$ (that Alice indexed separately).

Encapsulated Search Index. In order to solve the above motivating application, we will introduce a new primitive, which we term *Encapsulated Search Index* (ESI). As we will illustrate in Section 1.3, ESI is different than previously studied primitives in the area of searchable encryption. But for now, we informally summarize the main functionality and security properties of ESI (see also Definition 1):

- Phone can generate secret key $SK$, and send public key $PK$ to the Desktop.
- Given $PK$ and document $D$, Desktop can build an encrypted index $E$ for $D$, and a "compact" handle $c$.
- $D$ is then encrypted and erased (together with any local randomness created during the process), and Desktop only remembers $E, c$ and $PK$.
- Desktop can ask the Phone's permission to search for keyword $w$ in $D$, by sending it $w$ and the compact handle $c$.
- If approved, the Phone will use the secret key $SK$ to grant token $z = z(w, c, SK)$ to the Desktop.
- The Phone does not learn anything beyond $w$ from the handle $c$. This should hold *information-theoretically*.
- The Desktop can verify that the token $z$ indeed corresponds to $w$, and, if so, use $E, c, z$ and $PK$ to correctly learn if $w \in D$. In particular, the Phone cannot cause the Desktop to output a wrong answer (beyond denial of service).
- The token $z$ is specific to the pair $(D, w)$: the Desktop does not learn if other keywords $w' \neq w$ belong to $D$, or if $w$ belongs to other, freshly indexed documents $D'$.
- While each tuple $(E, c)$ is specific to the document $D$, the same $(PK, SK)$ pair should work for future documents $D'$, without compromising security.

[1] In fact, our solution will allow for generating secure indices even outside the Desktop, possibly by different parties. But for simplicity, we discuss the already interesting setting where Alice herself generates indices on the Desktop.

**Remark 1.** *For simplicity, we had the Desktop serve the role of both index creator and the storage location with the Phone serving the role of the search approver. However, the same could be generalized to the setting where the storage location is a company server, a trusted Desktop is the index creator, and the Phone is the search approver — all three being different parties.*

Additionally, in a good ESI, the overall search by the Desktop is much faster than the number of keywords in $D$. In fact, ideally, the bulk of the search should be done by the Desktop using any *non-private* dictionary structure, while the interaction between the Phone and the Desktop should have constant size/complexity, independent of $|D|$. Our main construction will have this property.

Extensions of ESI. For applications, we would also like to consider various extensions of ESI.

First, to mitigate Alice's worry that her Phone might be compromised, she might want to use a secure indexing scheme that is "friendly" to distributed implementation. For example, she might wish to secretly share her master key between her Phone, Laptop, and iPad (which we call *mobile devices* to differentiate them from the Desktop) in a way that she gets the token whenever two of them approve her search request. Moreover, this process should be *non-interactive*. The Desktop will send a request "Do you authorize to search document $D$ for keyword $w$?" to each of the $n$ mobile devices, and gets the token $z_w$ the moment $t \leq n$ of them respond affirmatively. Moreover, the Desktop can separately verify the authenticity of each of the shares from the mobile devices (which is why it does not need to wait for all $n$ to respond). The resulting notion of *threshold ESI* is formalized in Definition 2. This would correspond to the setting of multiple devices serving the role of the search approver.

Second, Alice might wish to delegate her searching ability to another user Bob, without the need to re-index the document. (A special case of this scenario is Bob being "Alice with a new Phone".) In this case, Alice does not want to freshly re-index the document, meaning that the encrypted index $E$ should not change. Instead, she only wants to convert the compact "handle" $c$ corresponding to her $PK$ to a new compact handle $c'$ corresponding to Bob's public key $PK'$. Once this conversion is done, Bob can use the pair $(E, c')$ with his Phone to search for keywords in the same document $D$. We formalize several flavors of such *delegatable ESI* in Definition 3.

Finally, we might want to have the ability to update the index $E$ by adding and deleting the keyword. In an *updatable ESI*, formalized in Definition 4, the token $z_w$ sent by Phone is also sufficient for the Desktop to update $E$ to $E'$ accordingly: remove, $w$ if $w$ was in $D$, or add it if it was not. This does not affect the handle $c$.

### 1.1 Our Main Tool: Encapsulated Verifiable Random Function

Naive Solution. Before introducing our solution approach, it is helpful to start with the naive solution which almost works. The Phone can generate a $(PK, SK)$ pair for a chosen-ciphertext-attack (CCA) secure encryption scheme. To index a document $D$, the Desktop can choose a seed $k$ for a pseudorandom function (PRF) $F_k$, and generate a standard (non-private) index $E$ by replacing each keyword $w \in D$ with the PRF value $y = F_k(w)$. These values are pseudorandom (hence, also distinct w.h.p.); thus, index $E$ will not reveal any information about $D$ except the number of keywords $N$.

The Desktop will finally generate a ciphertext $c$ encrypting $k$ under $PK$, and then erase the PRF key $k$. To get token for keyword $w$, the Desktop will send the tuple $(c, w)$ to the Phone, which will decrypt $c$ to get $k$, and return $y = F_k(w)$.

This naive solution satisfies our efficiency property and almost all the security properties. For example, the value $c$ is independent of the document $D$, so the Phone does not learn anything about the document (including search results). Similarly, the Desktop cannot use the token $y$ to learn about other keyword $w'$, as $y' = F_k(w')$ is pseudorandom given $y = F_k(w)$. The only basic property missing is verifiability: the Desktop cannot tell if the value $y$ indeed corresponds to $w$. This can be fixed by replacing PRF $F_K$ with a *verifiable random function* (VRF) [38]. A VRF has its own public-secret key pair $(pk, sk)$. For each input $w$, the owner of $sk$ can produce not only the function value $y = F_{sk}(w)$, but also a "proof" $z = z(sk, w)$. This proof can convince the verifier (who only knows $pk$) that the value $y$ is correct, while still leaving other yet "unproven" output $y' = F_{sk}(w')$ pseudorandom. While initial treatment of VRF focused on the "standard model" constructions [23, 24, 37, 38], VRFs are quite efficient in the random oracle model. In particular, several such efficient constructions are given the CFRG VRF standard [31, 32].

Deficiencies of the Naive Solution. While the composition of VRF and CCA encryption indeed works for the most basic ESI notion — and shows that *sublinear search can be meaningfully combined with public indexing*[2] — it seems too inflexible for our two main extensions: threshold ESI and delegatable ESI.

For threshold ESI, achieving "decrypt-then-evaluate-VRF" functionality *non-interactively* appears quite challenging with the current state-of-the-art. In particular, a natural way to accomplish this task would be to combine some non-interactive threshold CCA-decryption with a non-interactive threshold VRF implementation. Each of these advanced primitives is highly non-trivial but exists in isolation. For example, the works of [7, 12] show how to achieve non-interactive CCA-secure decryption in bilinear map groups. Unfortunately (for our purposes), both of these constructions encrypt elements of the "target bilinear group" $\mathbb{G}_1$ (see Section B). Thus, to get a non-interactive threshold ESI scheme we will need to build a non-interactive threshold VRF in which the secret key resides in the bilinear target group $\mathbb{G}_1$. No such construction is known, however. In fact, we are aware of only two recent non-interactive threshold VRF schemes, both proposed by [28].[3] Unfortunately, both of these constructions have the secret key over the standard group $\mathbb{Z}_p$, and cannot be composed with the schemes of [7, 12]. Hence, we either need to build a new (threshold) VRF with secret keys residing in $\mathbb{G}_1$, or build a new, *non-interactive*[4] threshold CCA decryption with keys residing in $\mathbb{Z}_p$. Both options seem challenging.

For delegatable ESI, our definitions (and the overall application) require an efficient procedure S-Check$(PK_1, c_1, PK_2, c_2)$ to check that the new handle $c_2$ was indeed delegated from $c_1$. The naive delegation scheme of decrypting $c_1$ to get VRF key $sk$, and then re-encrypting $sk$ with $PK_2$ does not have such efficient verifiability. We could try to attach a non-interactive zero-knowledge (NIZK) proof for this purpose, but such proof might be quite inefficient, especially with chosen *ciphertext* secure encryptions $c_1$ and $c_2$.

Our New Tool: Encapsulated VRF. Instead of tying our hands with the very specific and inflexible "CCA-encrypt-VRF-key" solution, we introduce a general primitive we call *encapsulated VRF* (EVRF). This primitive abstracts the core of the naive solution, but without insisting on a particular implementation. Intuitively, an EVRF allows the Phone to publish a public key $PK$, keep secret key $SK$ private so that the Desktop can use $PK$ to produce a ciphertext $C$ and trapdoor

[2] ESI is the first searchable encryption primitive to do so; see Section 1.3.

[3] As other prior distributed VRFs were either interactive [23, 36], or had no verifiability [2, 39] or offered no formal model/analysis [16, 17, 21, 35, 44].

[4] E.g., we cannot use the interactive threshold Cramer-Shoup [19] construction of [13].

key $T$ in a way that for any input $w$, the correct VRF value $y$ on $w$ can be efficiently evaluated in two different ways:

(a) Phone: using secret key $SK$ and ciphertext $C$.

(b) Desktop: using trapdoor $T$.

In addition, if the Desktop erased $T$ and only remembers $C$, $PK$, and $w$:

(c) Phone can produce a proof $z$ convincing Desktop that the value $y$ is correct.

(d) Without such proof, the value $y$ will look pseudorandom to the Desktop.

These properties are formalized in Definition 5. It is then easy to see that we can combine any EVRF with a non-private dictionary data structure, by simply replacing each keyword $w$ with EVRF output $y$, just as in the naive solution. See Construction 1. Moreover, this construction is very friendly to all our extensions. If the EVRF is a threshold (resp. delegatable) — see Definitions 8,10, — then we get threshold (resp. delegatable) ESI. Similarly, if the non-private data structure allows updates, our ESI construction is updatable.

To summarize, to efficiently solve all the variants of our Encapsulated Search Index scenario, we just need to build a *custom* EVRF which overcomes the difficulties we faced with the naive composition of VRF and CCA encryption.

### 1.2 Our EVRF Constructions

This is precisely what we accomplish: we build a simple and efficient EVRF under the Bilinear Decisional Diffie-Hellman (BDDH) assumption [9], in the random oracle model. Our basic EVRF is given in Construction 2. It draws a lot of inspiration and resemblance to the original Boneh-Franklin IBE (BF-IBE) [9], but with a couple of important tweaks. In essence, we observe that BF-IBE key encapsulation produces the ciphertext $R = g^r$ which is independent of the "target identity". Hence, we can use this value $R$ as "part of identity" $\mathsf{ID} = (R, w)$, where $w$ is our input/keyword, and still have a meaningful "ID-based secret key" $z_w$ corresponding to this identity. On the usability level, this trick allows the index generator to produce the value $R = g^r$ before any of subsequent EVRF inputs (keywords in our application) $w$ will be known. On a technical level, it allows us to "upgrade" BF-IBE from a chosen-plaintext attack (CPA) to CCA security for free.

Additionally, in Section 5.2 we show that our VRF construction easily lends itself to very simple, non-interactive threshold EVRF (which gives threshold ESI), by using Shamir's Secret Sharing [46], Feldman VSS [26], and the fact that the correctness of all computations is easily verified using the pairing. The resulting $(t, n)$-threshold implementation, given in Construction 3, is the best possible: it is non-interactive and every share is individually verifiable, which allows computing the output the moment $t$ correct shares are obtained.

Finally, Sections 6.2,6.3,6.4 extend our basic EVRF to various levels of delegatable EVRFs (which yield corresponding delegatable ESIs). All our constructions have a very simple delegation procedure, including a simple "equivalence" check to test if two handles correspond to the same EVRF under two different keys (which was challenging in the naive construction). The most basic delegatable EVRF in Sections 6.2 (Construction 4) is shown secure under the same BDDH assumption as the underlying EVRF. It assumes that all delegations are performed by non-compromised devices.

To handle delegation to/from an untrusted device, we modify our underlying EVRF construction to also include "BLS Signature" [11], to ensure that the sender "knew" the value $r$ used to generate the original handle $R = g^r$. See Section 6.3 and Construction 5. This new construction is shown to have "unidirectional" delegation security under the same BDDH assumption. Finally, we show that the same construction can be shown to satisfy even stronger levels of "bidirectional" delegation security, albeit under slightly stronger variants of BDDH we justify in the generic group model (see Sections 6.3,6.4 and Appendix B.1).

### 1.3 ESI vs Other Searchable Encryption Primitives

The notion of ESI is closely related to other searchable encryption primitives: most notably, *Searchable Symmetric Encryption* (SSE) [5, 15, 20, 22, 22, 30] and *Public-Key Encrypted Keyword Search* (PEKS) [1, 4, 8, 10, 43, 49]. Just like ESI, SSE and PEKS achieve the most basic property of any searchable encryption scheme, which we call *index privacy*: knowledge of encrypted index $E$ and several tokens $z_w$ does not reveal information about keywords $w'$ for which no tokens were yet given. I.e., the keywords in the index that have not been searched so far continue to remain private. Otherwise, the SSE/PEKS primitives have some notable differences from ESI. We discuss them below, simultaneously arguing why SSE/PEKS does not suffice for our application.

Setting of SSE. As suggested by its name, in this setting the index creator is the same party as the search approver, meaning that both parties must know the secret key $SK$ which is hidden from the Desktop storing the index. On the positive, this restriction allows for some additional properties which are hard or even impossible in the public-indexing setting of the ESI (and PEKS; see below). First, they allow for "universal searching", where the search approver can produce the token $z_w$ without getting the document-specific handle $c$: such token allows to simultaneously search different indices $E_1, E_2, \ldots$ corresponding to different documents $D_1, D_2, \ldots$.[5]

Second, one can talk about so-called "hidden queries" [22] which essentially captures the idea of "keyword-privacy". Specifically, the adversary who knows the index $E$ and keyword token $z$ should not learn if $z$ corresponds to keywords $w_0$ or keyword $w_1$.[6] With public-key indexing, such a strong semantic-security guarantee is impossible, at least when combined with universal searching: the adversary can always generate the index for some document $D_0$ containing $w_0$ and not $w_1$ and then test if $z$ works on this index.

We notice that "keyword privacy" and universal searching are not important for our motivating application. In fact, $w$ is generated by Alice when using the Desktop (and will be erased when no longer relevant). Moreover, our verifiability property of the ESI explicitly requires that the Desktop can check that the token $z_w$ is correct, explicitly at odds with keyword privacy. Additionally, when Alice sees the prompt on her phone asking if it is OK to search for the keyword $w$, she generally wants to know in what context (i.e., to what document $D$) this search would apply; and will not want a compromised token $z_w$ to search a more sensitive document $D'$. Thus, we do not insist on universal searching either in the ESI setting.

[5] From an application perspective, universal and document-specific setting are incomparable, as some application might want to restrict which keywords are allowed for different databases. On a technical level, however, a universal scheme can always be converted to a document-specific one, by prefixing the keyword with the name of the document $D$. Thus, universal searching is more powerful.

[6] Unfortunately, as surveyed by Cash *et al.* [14] and further studied by [20,34] (and others), *all* SSE schemes in the literature do not achieve the strongest possible keyword privacy and suffer from various forms of information leakage.

On the other hand, the biggest limitation of SSE — the inability to perform public-key indexing, — makes it inapplicable to our motivating application. First, at the time of index creation, Alice already has the entire document $D$ she wants to index on the Desktop, and she does not want to transmit this gigantic document to the Phone, have the Phone spend hours indexing it (or possibly run out of memory doing so), and then send the (also gigantic) index back to the Desktop. Second, even if efficiency was not an issue, Alice is not willing to fully trust her Phone either. For example, while Alice hopes that the Phone is more secure than the Desktop, it might be possible that the Phone is compromised as well. In this case, Alice wants the (compromised) Phone to only learn which keywords $w$ she is searching for, but not to learn anything else about the document $D$ (including if her searches were successful!). Moreover, even if Alice had a secure channel between the Desktop and the Phone, she does not want to use SSE and send-then-erase the corresponding secret key. Indeed, this method requires the phone to store a separate secret key for each document and also does not allow other parties to generate encrypted indexes for different files — a convenient feature Alice might find handy in the future.

To sum up, Alice wants to generate the entire encrypted index $E$ on her Desktop (and then erase/encrypt the document $D$), without talking to the Phone, and only contact the Phone to help authorize subsequent keyword searches. This means that SSE is inapplicable, and we must use public-key cryptography.

Setting of PEKS. In a different vein, PEKS allows Alice to publish a public-key $PK$ allowing anybody to create her encrypted index. Akin to SSE, PEKS also demand universal searching, meaning that the token $z_w$ can be produced independently of the (handle $c$ for the) document $D$. This means that strong keyword privacy is impossible (and, thus, not required) in PEKS.

More significantly for our purposes, this feature makes searching *inherently slow*: not as an artifact of the existing PEKS scheme, but as already mandated even by the *syntax* of PEKS. Specifically, to achieve universality, the index is created by indexing each keyword $w' \in D$ one-by-one (using $PK$), and then the token $z_w$ can only be used to test each such "ciphertext" $e$ separately, to see whether or not it corresponds to $w' = w$. Thus, inherently slow searching makes PEKS inapplicable as well for our motivating application. In contrast, the searching in the ESI is (required to be!) document-specific. As a result, we will be able to achieve the sublinear searching we desire.

Summary Comparison. Summarizing the above discussion (see Table 1), we can highlight five key properties of a given searchable encryption scheme: public-key indexing, sublinear search, universal search, keyword privacy, and index privacy. All of ESI/SSE/PEKS satisfy (appropriate form) of index privacy, and differ — sometimes by choice (ESI) or necessity (PEKS) — in terms of keyword privacy. So the most interesting three dimensions separating them are public-key indexing, sublinear search, and universal search, where (roughly) each primitive achieves two out of three. For our purposes, however, *ESI is the first primitive which combines public-key indexing and sublinear search*, which is precisely the setting of our motivating example. This forms the backbone of the commercially deployed product called Atakama [3].

## 2 Preliminaries

Notation. In this paper, we let $k$ be a security parameter. We employ the standard cryptographic model in which protocol participants are modeled by probabilistic polynomial (in $k$) time Turing

Table 1: A comparison of SSE, PEKS, and ESI.

| | SSE | PEKS | ESI |
|---|---|---|---|
| Public-Key Indexing | ✗ | ✓ | ✓ |
| Sublinear Search | ✓ | ✗ | ✓ |
| Universal Index | ✓ | ✓ | ✗ |
| Index Privacy | ✓ | ✓ | ✓ |
| Keyword Privacy | ✓ (partial) | ✗ (impossible) | ✗ (by choice!) |

machines (PPTs). We use $\text{poly}(k)$ to denote a polynomial function, and $\text{negl}(k)$ to refer to a negligible function in the security parameter $k$. For a distribution $X$, we use $x \leftarrow X$ to denote that $x$ is a random sample drawn from distribution $X$. For a set $S$ we use $x \leftarrow S$ to denote that $x$ is chosen uniformly at random from the set $S$. Additionally, we use the equality operator to denote a deterministic algorithm, and the $\rightarrow, \leftarrow$ operation to indicate a randomized algorithm.

Further, our EVRF constructions will use some "cryptographic hash function(s)" $H, H' : \{0,1\}^* \rightarrow \mathbb{G}$ mapping arbitrary-length strings (denoted $\{0,1\}^*$) to elements of the bilinear group $\mathbb{G}$. We produce a formal discussion about bilinear groups in Section B. The key property of these groups are that: for all $u, v \in \mathbb{G}$ and $x, y \in \mathbb{Z}$, we have $e(u^x, v^y) = e(u, v)^{xy}$. In our security proofs, where we reduce EVRF security to an appropriate assumption from Section B, we model the cryptographic hash functions as random oracles.

# 3 Encapsulated Search Index

We begin by formally introducing the new primitive of *standard* Encapsulated Search Index in Section 3.1, defining its syntax and security. We then present extensions to this primitive, adding features such as distribution (section 3.3), delegatability (section 3.4), and updatability (section 3.5).

## 3.1 Standard Encapsulated Search Index

We discussed, at length, the motivating application or setting for the primitive we call as Encapsulated Search Index in Section 1.
For visual simplicity, for the remainder of this section we will use upper-case letters ($D, E, Y$, etc.) to denote objects whose size can depend on the size of document $D$ (with the exception of various keys $SK, PK$, etc.), and by lower-case letters ($c, s, r, z, w$, etc.) objects whose size is constant.

In the definition below, we let $k$ be a security parameter, PPT stand for probabilistic polynomial-time Turing machines, $\text{poly}(k)$ to denote a polynomial function, and $\text{negl}(k)$ to refer to a negligible function in the security parameter $k$.

**Definition 1.** *An Encapsulated Search Index (ESI) is a tuple of PPT algorithms* $\text{ESI} = (\text{KGen}, \text{Prep}, \text{Index}, \text{S-S}$ *such that:*

- $\text{KGen}(1^k) \rightarrow (PK, SK)$*: outputs the public/secret key pair.*
- $\text{Prep}(PK) \rightarrow (s, c)$*: outputs compact representation* $c$*, and trapdoor* $s$*.*
- $\text{Index}(s, D) = E$*: outputs the encrypted index* $E$ *for a document* $D$ *using the trapdoor* $s$*.*
- $\text{S-Split}(PK, c') = r'$*: outputs a handle* $r'$ *from the representation* $c'$*.*

– S-$\textsc{Core}(SK, r', w) = z'$: *outputs a partial result* $z'$ *from the handle* $r'$.

– $\textsc{Finalize}(PK, E', c', z', w) = \beta \in \{0, 1, \perp\}$: *outputs* 1 *if the word* $w$ *is present in the original document* $D$, 0 *if not present, and* $\perp$ *if the partial output* $z'$ *is inconsistent.*

*Before we define the security properties, it is useful to define the following shorthand functions:*

– $\textsc{BldIdx}(PK, D) = (\textsc{Index}(s, D), c)$, *where* $(s, c) \leftarrow \textsc{Prep}(PK)$.

– S-$\textsc{Prove}(PK, SK, c, w) =$ S-$\textsc{Core}(SK,$ S-$\textsc{Split}(PK, c), w)$.

– $\textsc{Search}(PK, SK, (E, c), w) = \textsc{Finalize}(PK, E, c,$ S-$\textsc{Prove}(SK, c, w), w)$.

*We require the following security properties from this primitive:*

1. ***Correctness****: with prob.* 1 *(resp.* $(1 - negl(k))$*) over randomness of* KGen *and* Prep*, for all documents* $D$ *and keywords* $w \in D$ *(resp.* $w \notin D$*):*

$$\textsc{Search}(PK, SK, \textsc{BldIdx}(PK, D), w) = \begin{cases} 1 & \text{if } w \in D \\ 0 & \text{if } w \notin D \end{cases}$$

2. ***Uniqueness****: there exist no values* $(PK, E, c, z_1, z_2, w)$ *such that* $b_1 \neq \perp$, $b_2 \neq \perp$ *and* $b_1 \neq b_2$*, where:*

$$b_1 = \textsc{Finalize}(PK, E, c, z_1, w); \quad b_2 = \textsc{Finalize}(PK, E, c, z_2, w)$$

3. ***CCA Security****: We require that for any PPT algorithm* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *the following holds, where* $\mathcal{A}$ *does not make the query* S-$\textsc{Prove}(PK, SK, c^*, w)$ *with* $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$ *and* $|D_1| = |D_2|$*, for variables* $SK, c^*, D_1, D_2, w$ *defined below:*

$$\Pr\left[ b = b' \;\middle|\; \begin{array}{c} (PK, SK) \leftarrow \textsc{KGen}(1^k); \\ (D_1, D_2, st) \leftarrow \mathcal{A}_1^{\text{S-}\textsc{Prove}(PK, SK, \cdot, \cdot)}(PK); \\ b \leftarrow \{0, 1\}; \\ (E^*, c^*) \leftarrow \textsc{BldIdx}(PK, D_b); \\ b' \leftarrow \mathcal{A}_2^{\text{S-}\textsc{Prove}(PK, SK, \cdot, \cdot)}(E^*, c^*, st) \end{array} \right] \leq \frac{1}{2} + negl(k)$$

4. ***Privacy-Preserving*** [7]*: We require that for any PPT Algorithm* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *which outputs documents* $D_1, D_2$ *such that* $|D_1| = |D_2|$ *for variables* $D_1, D_2$ *defined below, the following holds:*

$$\Pr\left[ b = b' \;\middle|\; \begin{array}{c} (PK, SK) \leftarrow \textsc{KGen}(1^k); \\ (D_1, D_2, st) \leftarrow \mathcal{A}_1(PK, SK); \\ b \leftarrow \{0, 1\}; \\ (E^*, c^*) \leftarrow \textsc{BldIdx}(PK, D_b); \\ b' \leftarrow \mathcal{A}_2(c^*, st) \end{array} \right] \leq \frac{1}{2} + negl(k)$$

[7] It is easy to see that our syntax guarantees that any ESI construction is *unconditionally* **Privacy-Preserving** (even with knowledge of $SK$), for the simple reason that Prep that produces $c$ does not depend on the input document $D$. Thus, we will never explicitly address this property, but list it for completeness, as it is important for our motivating application.

**Remark 2.** *We want to ensure that an honest representation $c_1$ will not collide with another honest representation $c_2$. With this, we can ensure that honestly generated documents do not conflict. If there is a non-trivial chance of such a collision, then one can simply generate $c_2$ until collision with the challenge $c_1$. With this collision, and with knowledge of trapdoor $T_2$, one can trivially break security.*

**Remark 3.** *For efficiency, we will want* Search *to run in time $O(\log N)$ or less, where $N$ is the size of the document $D$. In fact, our main construction will have* S-Prove *run in time $O(1)$, independent of the size of the document, and* Finalize *would run in time at most $O(\log N)$, depending on the non-cryptographic data structure we use.*

### 3.2 Extensions to ESI

Threshold ESI. We extend the definition of the standard Encapsulated Search Index to achieve support for distributed token generation. To do this, we introduce a new algorithm called KG-Verify that aims to verify if the output of the KGen algorithm is correct, and replace Finalize with two more fined-grained procedures S-Verify and S-Combine. The formal discussion about the syntax and security of this primitive can be found in Section 3.3.

Delegatable ESI. We can also extend the definition of the standard Encapsulated Search Index to achieve support for delegation. Informally, Encapsulated Search Index is delegatable if there are two polynomial-time procedures S-Del, S-Check that work as follows: S-Del that achieves the delegation wherein it takes as input a representation $c$ corresponding to one key pair and produces a representation $c'$ corresponding to another key pair; S-Check helps verify if a delegation was performed correctly. The formal discussion about the syntax and security of this primitive, including several definitional subtleties, can be found in Section 3.4.

Updatable ESI. We can further extend the definition of the standard Encapsulated Search Index to support a use-case where one might want to remove a word, or add a word to the document $D$, without having to necessarily recompute the entire index. To achieve this, we need an additional algorithm called Update that can produce a new index $E'$ after performing an `action` relating to word $w$ in original index $E$, using the same token $z_w$ used for searching. The formal discussion about the syntax and security of this primitive can be found in Section 3.5.

### 3.3 Threshold Encapsulated Search Index

**Definition 2.** *A $(t, n)$-threshold Encapsulated Search Index (TESI) is a tuple of PPT algorithms* TESI = (KGen, KG-Verify, Prep, Index, S-Split, S-Core, S-Verify, S-Combine) *such that:*

- $\text{KGen}(1^k, t, n) \to (PK, \mathbf{SK} = (sk_1, \ldots, sk_n), \mathbf{VK} = (vk_1, \ldots, vk_n))$*: outputs the public key $PK$, a vector of secret shares $\mathbf{SK}$, and public key $PK$.*
- $\text{KG-Verify}(PK, \mathbf{VK}) = \beta \in \{0, 1\}$*: verifies that the output of* KGen *is indeed valid.*
- $\text{Prep}(PK) \to (s, c)$*: outputs compact representation $e$, and trapdoor $s$.*
- $\text{Index}(s, D) = E$*: outputs the encrypted index $E$ for a document $D$ using the trapdoor $T$.*
- $\text{S-Split}(PK, n, c') = (r'_1, \ldots, r'_n)$*: outputs $n$ handles $r'_1, \ldots, r'_n$ from $c'$.*

– S-$\textsc{Core}(sk_i, r'_i, w) = z'_i$: *outputs partial result on input $x$ using handle $r'_i$ and secret key share $sk_i$.*

– S-$\textsc{Verify}(PK, vk_i, z'_i, w) = \beta \in \{0,1\}$: *verifies that the share $z'_i$ produced by party $i$ is valid.*

– S-$\textsc{Combine}(PK, E', c', z'_{i_1}, \ldots, z'_{i_t}, w) = \beta \in \{0,1\}$: *uses the partial shares to determine if the word $w$ is present in $D$ or not* [8].

*Before we define the security properties, it is useful to define the following shorthand functions:*

– $\textsc{BldIdx}(PK, D) = (\textsc{Index}(s, D), c)$ *where* $(s, c) \leftarrow \textsc{Prep}(PK)$.

– S-$\textsc{Prove}(\mathbf{SK}, i, c, w) =$ S-$\textsc{Core}(sk_i, r_i, w)$ *where* $r_1, \ldots, r_n =$ S-$\textsc{Split}(PK, n, c)$

– $\textsc{Search}(\mathbf{SK}, i_1, \ldots, i_t, (E, c), w)$: *For $j = 1, \ldots, t$ let* $z_{i_j} =$ S-$\textsc{Prove}(\mathbf{SK}, i, c, w)$.
*Output $\perp$ if, for some $1 \le j \le t$,* S-$\textsc{Verify}(PK, vk_{i_j}, z_{i_j}, w) = 0$.
*Otherwise, output* S-$\textsc{Combine}(PK, E, c, z_{i_1}, \ldots, z_{i_t}, w)$.

*We require the following security properties from this primitive:*

1. ***Correctness**:*

   *(a) with prob. 1 over randomness of* $\text{KG}\textsc{en}(1^k, t, n) \to (PK, \mathbf{SK}, \mathbf{VK})$,
   $\text{KG-}\textsc{Verify}(PK, \mathbf{VK}) = 1$.

   *(b) with prob. 1 (resp. $(1 - negl(k))$) over randomness of* KGen *and* Prep*, for all documents $D$ and keywords $w \in D$ (resp. $w \notin D$):*

$$\textsc{Search}(\mathbf{SK}, i_1, \ldots, i_t, \textsc{BldIdx}(PK, D), w) = \begin{cases} 1 & w \in \text{ if } D \\ 0 & w \text{ if } \notin D \end{cases}$$

2. ***Uniqueness**: there exist no values $(PK, \mathbf{VK}, E, c, Z_1, Z_2, w)$ where $Z_1 = ((i_1, z_{i_1}), \ldots, (i_t, z_{i_t}))$ and $Z_2 = ((j_1, z_{j_1}), \ldots, (j_t, z_{j_t}))$. s.t.*

   *(a)* $\textsc{Gen-Vfy}(PK, \mathbf{VK}) = 1$,

   *(b) for $k = 1, \ldots, t$:*
   – S-$\textsc{Verify}(PK, vk_{i_k}, z_{i_k}, w) = 1$.
   – S-$\textsc{Verify}(PK, vk_{j_k}, z_{j_k}, w) = 1$,.

   *(c) and*

$$\text{S-}\textsc{Combine}(PK, E, c, z_{i_1}, \ldots, z_{i_t}, w) \neq \text{S-}\textsc{Combine}(PK, E, c, z_{j_1}, \ldots, z_{j_t}, w)$$

3. ***CCA Security**: We require that for any PPT algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ the following holds, where $\mathcal{A}$ does not make the query* S-$\textsc{Prove}(\mathbf{SK}, j, c^*, w)$ *with $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$, $|D_1| = |D_2|$ and $j \notin \{i_1, \ldots, i_{t-1}\}$, for variables $\mathbf{SK}, i_1, \ldots, i_{t-1}, c^*, D_1, D_2$ defined below:*

[8] Without loss of generality, we will always assume that all the $t$ partial evaluations $z_i$ satisfy S-$\textsc{Verify}(PK, vk_i, z'_i) = 1$ (else, we output $\perp$ before calling S-Combine).

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \{i_1, \ldots, i_{t-1}, st\} \leftarrow \mathcal{A}_0(1^k, t, n) \\ (PK, \mathbf{SK}, \mathbf{VK}) \leftarrow \text{KGen}(1^k, t, n); \\ (D_1, D_2, st) \leftarrow \mathcal{A}_1^{\text{S-Prove}(\mathbf{SK}, \cdot, \cdot, \cdot)}(PK, \mathbf{VK}, \mathbf{SK}', st); \\ b \leftarrow \{0,1\}; (E^*, c^*) = \text{BldIdx}(PK, D_b); \\ b' \leftarrow \mathcal{A}_2^{\text{S-Prove}(\mathbf{SK}, \cdot, \cdot, \cdot)}(E^*, c^*, st) \end{array}\right] \leq \frac{1}{2} + negl(k)$$

*where* $\mathbf{SK}' = (sk_{i_1}, \ldots, sk_{i_{t-1}})$.

4. ***Privacy-Preserving****: We require that for any PPT Algorithm* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *which outputs documents* $D_1, D_2$ *such that* $|D_1| = |D_2|$ *for variables* $D_1, D_2$ *defined below, the following holds:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} (PK, \mathbf{SK}, \mathbf{VK}) \leftarrow \text{KGen}(1^k, t, n); \\ (D_1, D_2, st) \leftarrow \mathcal{A}_1(PK, \mathbf{SK}, \mathbf{VK}); \\ b \leftarrow \{0,1\}; \\ (c^*, E^*) = \text{BldIdx}(PK, D_b); \\ b' \leftarrow \mathcal{A}_2(c^*, st) \end{array}\right] \leq \frac{1}{2} + negl(k)$$

It is again easy to see that the above primitive is unconditionally **Privacy-Preserving**, even with knowledge of $\mathbf{SK}$, since Prep that produces $e$ does not depend on the input document $D$.

### 3.4 Delegatable Encapsulated Search Index

In this section, we extend the definition of the standard Encapsulated Search Index to achieve delegatability. Our definition will capture 3 security levels. Similar levels of security will also appear in the DEVRF definition (discussed in Section 6.1).

**Definition 3.** *An Encapsulated Search Index* $\text{ESI} = (\text{KGen}, \text{Prep}, \text{Index}, \text{S-Split}, \text{S-Core}, \text{Finalize})$ *is also delegatable if there exists polynomial-time procedure* S-Del, S-Check *such that:*

- $\text{S-Del}(SK_1, c_1, SK_2) = c_2$*: outputs a representation* $c_2$ *corresponding to key pair* $(PK_2, SK_2)$ *from a representation* $c_1$ *corresponding to key pairs* $(PK_1, SK_1)$.[9]
- $\text{S-Check}(PK_1, c_1, PK_2, c_2) = \beta \in \{0,1\}$*:*

*Before we define the security properties, it is useful to define the following shorthand functions:*

- $\text{BldIdx}(PK, D) = (\text{Index}(s, D), c)$ *where* $(s, c) \leftarrow \text{Prep}(PK)$.
- $\text{S-Prove}(SK, c, w) = \text{S-Core}(SK, \text{S-Split}(PK, c), w)$.
- $\text{Search}(SK, (E, c), w) = \text{Finalize}(PK, E, c, \text{S-Prove}(SK, c, w), w)$.

*In addition to the standard Encapsulated Search Index properties of* ***Correctness****,* ***Uniqueness****, and* ***Privacy-Preserving****, we require the following security properties from a delegatable Encapsulated Search Index:*

[9] By default, this algorithm takes as input the secret key of the other party. We can consider a publicly-delegatable algorithm that takes as input only the public key of the second party.

1. ***Delegation-Completeness**: for any valid $(PK_1, SK_1)$, $(PK_2, SK_2)$, and compact representation $c_1$*

$$\text{S-DEL}(SK_1, c_1, SK_2) = c_2 \implies \text{S-CHECK}(PK_1, c_1, PK_2, c_2) = 1$$

2. ***Delegation-Soundness**: for any valid $(PK_1, SK_1)$, $(PK_2, SK_2)$, encrypted index $E$, and compact representations $c_1, c_2$*

$$\text{S-CHECK}(PK_1, c_1, PK_2, c_2) = 1 \implies \\ \forall w \ \text{SEARCH}(SK_1, E, c_1, w) = \text{SEARCH}(SK_2, E, c_2, w)$$

*Before we define the **CCA** Security, it is useful to define the following set of oracles:*

- $\text{REG}(1^k)$*: registration oracle. Every call increments a global counter $q$, calls $(PK_q, SK_q) \leftarrow \text{KGEN}(1^k)$, records $(q, PK_q, SK_q)$, and returns $(q, PK_q)$ to the adversary.*
- $\text{HPROVE}(i, c, w)$*: honest evaluation oracle. Here, $1 \le i \le q$ and the oracle returns $\text{S-PROVE}(SK_i, c, w)$.*
- $\text{HDEL}(i, c, j)$*: honest delegation oracle. Here $1 \le i, j \le q$, and the oracle returns $c_2 = \text{S-DEL}(SK_i, c, SK_j)$*
- $\text{OUTDEL}(i, c, SK)$*:"out-delegation oracle". Here $1 \le i \le q$, and the oracle returns $e' = \text{S-DEL}(SK_i, c, SK)$.*
- $\text{INDEL}(SK, c, j)$*:"in-delegation oracle". Here $1 \le j \le q$, and the oracle returns $e' = \text{S-DEL}(SK, c, SK_j)$.*

3. ***CCA Security**: We require that for any PPT algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which outputs documents $D_1, D_2$ such that $|D_1| = |D_2|$ for variables $D_1, D_2$ defined below, and where legality of $\mathcal{A}$ and appropriate delegation oracle(s) are defined separately, the following holds:*

$$\Pr\left[\, b = b' \;\middle|\; \begin{array}{c} (1, PK_1) \leftarrow \text{REG}(1^k); \\ (D_1, D_2) \leftarrow \mathcal{A}_1^{\text{REG},\text{HPROVE},\mathcal{O}}(PK_1); \\ b \leftarrow \{0,1\}; \\ (E^*, c^*) = \text{BLDIDX}(PK, D_b); \\ b' \leftarrow \mathcal{A}_2^{\text{REG},\text{HPROVE},\mathcal{O}}(E^*, c^*) \end{array} \right] \le \frac{1}{2} + negl(k)$$

   (a) ***Basic CCA Security**: $\mathcal{O} = \{\text{HDEL}\}$.*
   *Legality of $\mathcal{A}$: no call to $\text{HPROVE}(i, c, w)$ s.t.*
   *$\text{S-CHECK}(PK_1, c^*, PK_i, c) = 1$ and $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$.*

   (b) ***Uni CCA Security**: $\mathcal{O} = \{\text{HDEL}, \text{OUTDEL}\}$.*
   *Legality of $\mathcal{A}$: no call to $\text{HPROVE}(i, c, w)$ or $\text{OUTDEL}(i, c, *)$ s.t.*
   *$\text{S-CHECK}(PK_1, c^*, PK_i, c) = 1$ and $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$.*

   (c) ***Bi CCA Security**: $\mathcal{O} = \{\text{HDEL}, \text{OUTDEL}, \text{INDEL}\}$.*
   *Legality of $\mathcal{A}$: no call to $\text{HPROVE}(i, c, w)$ or $\text{OUTDEL}(i, c, *)$ s.t.*
   *$\text{S-CHECK}(PK_1, c^*, PK_i, c) = 1$ and $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$.*

**Remark 4.** *It is easy to see that **Delegation-Completeness** and **Delegation-Soundness** imply **Delegation-Correctness** which is defined as follows: for any valid $(PK_1, SK_1)$, $(PK_2, SK_2)$, and compact representation $c_1$*

$$\text{S-DEL}(SK_1, c_1, SK_2) = c_2 \implies$$
$$\forall w \ \text{SEARCH}(SK_1, E, c_1, w) = \text{SEARCH}(SK_2, E, c_2, w)$$

### 3.5 Updatable Encapsulated Search Index

Another useful extension to Encapsulated Search Index would be to support operations that help update the index. However, note that the process of updating should not reveal information about the underlying keywords. To achieve this, we extend the standard definition.

**Definition 4.** *An Encapsulated Search Index* ESI = (KGEN, PREP, INDEX, S-SPLIT, S-CORE, FINALIZE) *is also updatable if there exists polynomial-time procedures* UPDATE *such that:*

- $\text{UPDATE}(E, c, w, z, \texttt{action}) \to E' \cup \perp$, *where* $\texttt{action} \in \{\texttt{add}, \texttt{remove}\}$.

*In addition to the standard Encapsulated Search Index properties of* ***Correctness****,* ***Uniqueness****,* ***CCA Security*** *and* ***Privacy-Preserving****, we require the following security properties from an updatable Encapsulated Search Index:*

***Update Correctness****: over the randomness of* KGEN, PREP, INDEX*, for any document* $D_0$*, update sequence* $(\texttt{action}_1, w_1), \ldots, (\texttt{action}_q, w_q)$ *and keyword* $w$*, the following holds with probability* $(1 - negl(k))$*:*

- *Let* $(c, E_0) \leftarrow \text{BLDIDX}(PK, D_0)$.
- *For* $i = 1$ *to* $q$*, let:*
  - $z_i = \text{S-PROVE}(SK, c, w_i)$;
  - $E_i = \text{UPDATE}(E_{i-1}, c, w_i, z_i, \texttt{action}_i)$;
  - $D_i$ *be correct update of* $D_{i-1}$ *following* $\texttt{action}_i$ *on* $w_i$.
- *Then* $\text{SEARCH}(SK, (E_q, c), w) = \begin{cases} 1 & \text{if } w \in D_q \\ 0 & \text{if } w \notin D_q \end{cases}$

## 4 Encapsulated Verifiable Random Functions (EVRFs)

As mentioned earlier, we use a new primitive called Encapsulated Verifiable Random Function to build the encapsulated search index. In this section, we begin by introducing this primitive in section 4.1. In Section 4.3, we present an overview of extensions to this primitive. Later sections in paper contained detailed expositions on the extensions.

### 4.1 Standard EVRFs

Intuitively, an EVRF allows the receiver Alice to publish a public key $PK$ and keep secret key $SK$ private so that any sender Bob can use $PK$ to produce a ciphertext $C$ and trapdoor key $T$ in a way such that for any input $x$, the correct VRF value $y$ on $x$ can be efficiently evaluated in two different ways:

(a) Alice can evaluate $y$ using secret key $SK$ and ciphertext $C$.

(b) Bob can evaluate $y$ using trapdoor $T$.

In addition, for any third party Charlie who knows $C$, $PK$ and $x$:

(c) Alice can produce a proof $z$ convincing Charlie that the value $y$ is correct.

(d) Without such proof, the value $y$ will look pseudorandom to Charlie.

**Definition 5.** *An* Encapsulated Verifiable Random Function *(EVRF) is a tuple of PPT algorithms* EVRF $= (\textsc{Gen}, \textsc{Encap}, \textsc{Comp}, \textsc{Split}, \textsc{Core}, \textsc{Post})$ *such that:*

- $\textsc{Gen}(1^k) \to (PK, SK)$*: outputs the public/secret key pair.*
- $\textsc{Encap}(PK) \to (C, T)$*: outputs ciphertext* $C$ *and trapdoor* $T$*.*
- $\textsc{Comp}(T, x) = y$*: evaluates EVRF on input* $x$*, using trapdoor* $T$*.*
- $\textsc{Split}(PK, C') = R'$*: outputs a handle from full ciphertext* $C'$*.*
  *Note, this preprocessing is independent of the input* $x$*, can depend on the public key* $PK$*, but* not *on the secret key* $SK$*.*[10]
- $\textsc{Core}(SK, R', x) = z'$*: evaluates partial EVRF output on input* $x$*, using the secret key* $SK$ *and handle* $R'$*.*
- $\textsc{Post}(PK, z', C', x) = y' \cup \perp$*: outputs either the EVRF output from the partial output* $z'$*, or* $\perp$*.*

*Before we define the security properties, it is useful to define the following shorthand functions:*

- $\textsc{Prove}(PK, SK, C, x) = \textsc{Core}(SK, \textsc{Split}(PK, C), x)$
- $\textsc{Eval}(PK, SK, C, x) = \textsc{Post}(PK, \textsc{Prove}(SK, C, x), C, x)$

*We require the following security properties:*

1. ***Evaluation-Correctness****: with prob.* 1 *over randomness of* Gen *and* Encap*, for honestly generated ciphertext* $C$ *and for all inputs* $x$*,*

$$\textsc{Comp}(T, x) = \textsc{Eval}(PK, SK, C, x)$$

2. ***Uniqueness****: there exist no values* $(PK, C, x, z_1, z_2)$ *s.t.* $y_1 \neq \perp, y_2 \neq \perp,$ *and* $y_1 \neq y_2$ *where*

$$y_1 = \textsc{Post}(PK, z_1, C, x), \quad y_2 = \textsc{Post}(PK, z_2, C, x)$$

[10] The algorithm Split is not technically needed, as one can always set $R = C$. In fact, this will be the case for our EVRF in section 4.4. However, one could envision EVRF constructions where the Split procedure can do a non-trivial (input-independent) part of the overall Prove = Core(Split) procedure, and without the need to know the secret key $SK$. This will be the case for some of the delegatable EVRFs we consider in Section 6.1.

3. ***Pseudorandomness under*** CORE ***(\$-Core)****: for any PPT algorithm* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, where* $\mathcal{A}$ *does not make query* $(C, x)$ *to* $\text{PROVE}(PK, SK, \cdot, \cdot)$*, for variables* $SK, C, x$ *defined below, the following holds:*

$$\Pr\left[ b = b' \;\middle|\; \begin{array}{c} (PK, SK) \leftarrow \text{GEN}(1^k); \\ (C, T) \leftarrow \text{ENCAP}(PK); \\ (x, st) \leftarrow \mathcal{A}_1^{\text{PROVE}(PK,SK,\cdot,\cdot)}(PK, C); \\ y_0 = \text{COMP}(T, x); \quad y_1 \leftarrow \{0,1\}^{|y_0|}; \\ b \leftarrow \{0,1\}; \quad b' \leftarrow \mathcal{A}_2^{\text{PROVE}(PK,SK,\cdot,\cdot)}(y_b, st) \end{array} \right] \leq \frac{1}{2} + negl(k)$$

We present a construction of our EVRF in section 4.4.

**Remark 5.** *We note that any valid ciphertext* $C$ *implicitly defines a standard verifiable random function (VRF). In particular, the value* $z = \text{PROVE}(SK, C, x)$ *could be viewed as the VRF proof, which is accepted iff* $\text{POST}(PK, z, C, x) \neq \perp$*.*

**Remark 6.** *We reiterate that our pseudorandomness definition does not give the attacker "unguarded" access to the* CORE *procedure, but only "*SPLIT*-guarded" access to* $\text{PROVE} = \text{CORE}(\text{SPLIT})$*. This difference does not matter when the* SPLIT *procedure just sets* $R = C$*. However, when* SPLIT *is non-trivial, the owner of* $SK$ *(Alice) can only outsource it to some outside server (Charlie) if it trusts Charlie and the authenticity (but not privacy) of the channel between Alice and Charlie.*

## 4.2 Generic Construction of Encapsulated Search Index

NON-PRIVATE DICTIONARY DATA STRUCTURE. Our generic construction will use the simplest kind of non-cryptographic dictionary which allows one to preprocess some set $D$ into some data structure $E$ so that membership queries $w \in D$ can be answered in sub-linear time in $N = |D|$. In particular, a classic instantiation of such a dictionary could be any balanced search trees with search time $O(\log N)$. If a small probability of error is allowed, we could also use faster data structures, such as hash tables [18], Bloom filters [6,40,41] or cuckoo hash [42], whose search takes expected time $O(1)$. The particular choice of the non-cryptographic dictionary will depend on the application, which is a nice luxury allowed by our generic composition.

Formally, a non-private dictionary $\text{DS} = (\text{CONSTRUCT}, \text{FIND})$ is any data structure supporting the following two operations:

- $\text{CONSTRUCT}(D) \to E$: outputs the index $E$ on an input document $D$.
- $\text{FIND}(E, w) \to \{0, 1\}$: outputs 1 if $w$ is present in $D$, and 0 otherwise. We assume perfect correctness for $w \in D$, and allow negligible error probability for $w \notin D$.

OUR COMPOSITION. We show that Encapsulated Search Index can be easily built from any such non-cryptographic dictionary $\text{DS} = (\text{CONSTRUCT}, \text{FIND})$ and and $\text{EVRF} = (\text{GEN}, \text{ENCAP}, \text{COMP}, \text{SPLIT}, \text{CORE}, \text{POST})$. This composition is given below in Construction 1.

EFFICIENCY. By design, the SEARCH operation of our composition inherits the efficiency of the non-cryptographic dictionary DS. In particular, it is $O(\log |D|)$ with standard balanced search trees and could become potentially $O(1)$ with probabilistic dictionaries, such as hash tables or Bloom filters.

**Protocol** Generic ESI Construction

KGen($1^k$)

Run EVRF.Gen($1^k$) $\to (PK, SK)$.
**return** $PK, SK$.

Prep($PK$)

Run EVRF.Encap($PK$) $\to (C, T)$.
**return** $c = C$ and $s = T$.

Index($s, D$)

**for** $w \in D$ **do**
Compute $y_w = \text{EVRF.Comp}(s, w)$.
Compute $Y = \{y_w | w \in D\}$.
Run DS.Construct($Y$) $\to E$.
**return** $E$.

S-Split($PK, c'$)

Run EVRF.Split($PK, c'$) $= r'$.
**return** $r'$.

S-Core($SK, r', w$)

Run EVRF.Core($SK, r', w$) $= z'$.
**return** $z'$.

Finalize($PK, E', c', z', w$)

Run EVRF.Post($PK, z', c', w$) $= y'$.
**if** $y' = \perp$ **then**
**return** $\perp$.
**else**
**return** DS.Find($E', y'$).

Construction 1: Generic ESI = (KGen, Prep, Index, S-Split, S-Core, Finalize).

Security Analysis. The **Correctness** and **Uniqueness** properties of the above construction trivially follows from the respective properties of the underlying EVRF and $DS$. In particular, we get negligible error probability for $w \notin D$ either due to unlikely EVRF collision between $y_w$ and $y_{w'}$ for some $w' \in D$, or a false positive of the $DS$. In Section C.1 we prove the following theorem:

**Theorem 6.** *If* EVRF *satisfies the* ***$-Core*** *property, then Encapsulated Search Index is* ***CCA*** *secure. Further, if the* EVRF *(resp.* DS *) is threshold and/or delegatable (resp. updatable; see Remark 7), the resulting ESI inherits the same.*

**Remark 7.** *It is easy to see that our construction of Encapsulated Search Index (Construction 1) can be made updatable if the underlying data structure supports the addition and removal. Formally the* DS *also has the following additional operation:*

- DS.Modify($E, w,$ `action`) $\to E'$*: adds/removes* $w$ *to/from the index* $E$ *when* `action` = `add`/`remove`*, and outputs the new index* $E'$ *.*

*Then, the* Update($E, c, w, z,$ `action`) *algorithm can be defined as follows:*

- *Compute* EVRF.Post($PK, z, c, w$) $= y$.
- *If* $y = \perp$*, return* $\perp$.
- *Otherwise, return* $E' =$ DS.Modify($E, y,$ `action`)

### 4.3 Extensions to EVRFs

The generic construction of ESI from Section 4.2 can be extended to achieve the various extensions of ESI, as defined in Section 3. We do this by extending the EVRF definitions, and instantiating each ESI with its corresponding EVRF and inheriting the required functionality.

Threshold EVRF. In the earlier definition, we had a single secret key $SK$. With possession of this secret key, one can evaluate the EVRF on any input $x$. Therefore, it becomes imperative to protect the key from leakage. Indeed, it is natural to extend our early definition to cater to the setting of a distributed evaluation of the EVRF. The key difference in the definition of threshold EVRF from the earlier definition is that the Post algorithm is now formally split into the share verification algorithm Shr-Vfy and the final evaluation algorithm Combine. The formal discussion about the syntax and security of this primitive can be found in Section 5.1.

**Remark 8.** *It is easy to see that our construction of Encapsulated Search Index (Construction 1) inherits the different properties of the underlying encapsulated verifiable random function. In other words, by using* TEVRF*, one can construct a threshold Encapsulated Search Index by suitably mapping the functions as follows:*

- KG-Verify$(PK, \mathbf{VK}) =$ Gen-Vfy$(PK, \mathbf{VK})$
- S-Verify$(sk_i, z'_i, w) =$ Shr-Vfy$(sk_i, z'_i, w)$
- S-Combine$(PK, E', c', z'_{i_1}, \ldots, z'_{i_t}, w)$*: run*

$$y' = \text{Combine}(PK, c', z'_{i_1}, \ldots, z'_{i_t}, w)$$

*and then output* DS.Find$(E', y')$

*Therefore, Theorem 6 can be extended to say that if* TEVRF *satisfies the* ***$-DCore*** *property, then we have a* ***CCA*** *secure threshold Encapsulated Search Index.*

Delegatable EVRF. Next, we extend the definition of *standard* EVRFs to the setting where the EVRF owner could delegate its evaluation power to another key. Recall that a standard EVRF has the following algorithms: Gen, Encap, Comp, Split, Core, Post. Delegation, therefore, implies that one can convert a ciphertext $C_1$ for key pair $(PK_1, SK_1)$ to ciphertext $C_2$ for a different key pair $(PK_2, SK_2)$ which encapsulates the same VRF, i.e.,

$$\forall x,\ \text{Eval}(PK_1, SK_1, C_1, x) = \text{Eval}(PK_2, SK_2, C_2, x) \tag{1}$$

where $\text{Eval}(PK, SK, C, x) = \text{Post}(PK, \text{Prove}(SK, c, x), C, x)$. The formal discussion about the syntax and security of this primitive can be found in Section 6.1.

**Remark 9.** *It is easy to see that our construction of Encapsulated Search Index (Construction 1) inherits the different properties of the underlying encapsulated verifiable random function. In other words, by using a* DEVRF*, one can construct a delegatable Encapsulated Search Index by suitably mapping the functions as follows:*

- S-Del$(SK_1, c_1, SK_2) =$ Del$(SK_1, c_1, SK_2)$
- S-Check$(PK_1, c_1, PK_2, c_2) =$ Same$(PK_1, c_1, PK_2, c_2)$

*In addition, the Encapsulated Search Index also achieves different levels of* **CCA** *security based on the security level of* ***$-Core*** *property of the delegatable EVRF.*

### 4.4 Standard EVRF

We now present the standard EVRF construction, presented in Construction 2.

**Protocol** Standard EVRF

$\underline{\text{Gen}(1^k)}$
Sample $a \in_r \mathbb{Z}_p^*$
Compute $A = g^a \in \mathbb{G}$.
**return** $SK = a$ and $PK = (g, A)$.

$\underline{\text{Encap}(PK)}$
Parse $PK = (g, A)$.
Sample $r \in_r \mathbb{Z}_p^*$.
Compute $R = g^r, S = A^r$.
**return** $C = R$, $T = (R, S)$.

$\underline{\text{Comp}(T, x)}$
Parse $T = (R, S)$.
Compute $y = e(H(R, x), S)$.
**return** $y$.

$\underline{\text{Split}(PK, C')}$
Parse $PK = (g, A)$, $C' = R'$.
**return** $R'$.

$\underline{\text{Core}(SK, C', x)}$
Parse $SK = a, C' = R'$.
Compute $z = H(R', x)^a$.
**return** $z$.

$\underline{\text{Post}(PK, z, C', x)}$
Parse $PK = (g, A)$, $C' = R'$
**if** $e(z, g) \neq e(H(R', x), A)$ **then**
  **return** $\perp$.
**else**
  Compute $y' = e(z, R')$.
  **return** $y'$.

Construction 2: Standard EVRF = (Gen, Encap, Comp, Split, Core, Post).

**Security Analysis.** To check **Evaluation-Correctness**, we observe that $A^r = g^{ar} = R^a$, and by the bilinearity we have:

$$\text{Comp}(T = (R, S), x) = e(H(R, x), S) = e(H(R, x), A^r)$$

From our earlier observation, we get that:

$$e(H(R, x), A^r) = e(H(R, x), R^a) = e(H(R, x)^a, R) = e(z, R)$$

This is the same as $\text{Post}(A, \text{Core}(a, \text{Split}(A, R), x), R, x)$ which concludes the proof.

To prove **Uniqueness**, consider any tuple $(PK = A, C = R, x, z_1, z_2)$. Further, let $y_1 = \text{Post}(A, z_1, R, x)$ and $y_2 = \text{Post}(A, z_2, R, x)$. If $y_1 \neq \perp$ and $y_2 \neq \perp$, then we have that $e(z_1, g) = e(H(R, x), A) = e(z_2, g)$. From definition of bilinear groups, we get that $z_1 = z_2$. Consequently, $y_1 = e(z_1, R) = e(z_2, R) = y_2$.

Finally, we prove the following result in Section C.2.

**Theorem 7.** *The standard* EVRF *given in Construction 2 satisfies the* ***$-Core*** *property under the* BDDH *assumption in the random oracle model.*

# 5 Threshold Encapsulated Verifiable Random Functions

In this section, we formally introduce the primitive known as a Threshold EVRF in Section 5.1. We then present a construction of Threshold EVRF in Section 5.2 but defer the security proof due to space constraints. The proof can be found in Section C.3.

## 5.1 Definition of Threshold (or Distributed) EVRFs

**Definition 8.** *A* $(t, n)$-Threshold EVRF *is a tuple of PPT algorithms* TEVRF = (Gen, Gen-Vfy, Encap, Comp, Split, D-Core, Shr-Vfy, Combine) *such that:*

- $\text{Gen}(1^k, t, n) \to (PK, \boldsymbol{SK} = (sk_1, \ldots, sk_n), \boldsymbol{VK} = (vk_1, \ldots, vk_n))$*: outputs the public key* $PK$*, a vector of secret shares* $\boldsymbol{SK}$*, and public shares* $\boldsymbol{VK}$*.*

- $\textsc{Gen-Vfy}(PK, \mathbf{VK}) = \beta \in \{0,1\}$: *verifies that the output of* $\textsc{Gen}$ *is indeed valid.*
- $\textsc{Encap}(PK) \to (C, T)$: *outputs ciphertext* $C$ *and trapdoor* $T$.
- $\textsc{Comp}(T, x) = y$: *evaluates EVRF on input* $x$*, using trapdoor* $T$.
- $\textsc{Split}(PK, n, C') = (R'_1, \ldots R'_n)$: *outputs* $n$ *handles* $R'_1, \ldots, R'_n$ *from full ciphertext* $C'$.
- $\text{D-}\textsc{Core}(sk_i, R'_i, x) = z'_i$: *evaluates EVRF share on input* $x$*, using handle* $R'_i$ *and secret key share* $sk_i$.
- $\textsc{Shr-Vfy}(PK, vk_i, z'_i, x) = \beta \in \{0,1\}$: *verifies that the share produced by the party* $i$ *is valid.*
- $\textsc{Combine}(PK, C', z'_{i_1}, \ldots, z'_{i_t}, x) = y'$: *uses the partial evaluations* $z'_{i_1}, \ldots, z'_{i_t}$ *to compute the final value of EVRF on input* $x$.[11]

*Before we define the security properties, it is useful to define the following shorthand functions:*

- $\textsc{Prove}(\mathbf{SK}, i, C, x) = \text{D-}\textsc{Core}(sk_i, R_i, x)$, *where* $(R_1, \ldots, R_n) = \textsc{Split}(PK, n, C)$.
- $\textsc{Eval}(\mathbf{SK}, i_1, \ldots, i_t, C, x)$: *For* $j = 1 \ldots t$, *compute* $z_{i_j} = \textsc{Prove}(\mathbf{SK}, i_j, C, x)$. *Output* $\perp$ *if, for some* $1 \le j \le t$, $\textsc{Shr-Vfy}(PK, vk_{i_j}, z_{i_j}, x) = 0$. *Otherwise, output* $\textsc{Combine}(PK, C, z_{i_1}, \ldots, z_{i_t}, x)$.

*We require the following security properties:*

1. ***Distribution-Correctness****:*
   (a) *with prob. 1 over randomness of* $\textsc{Gen}(1^k, t, n) \to (PK, \mathbf{SK}, \mathbf{VK})$, $\textsc{Gen-Vfy}(PK, \mathbf{VK}) = 1$
   (b) *with prob. 1 over randomness of* $\textsc{Gen}$ *and* $\textsc{Encap}$*, for honestly generated ciphertext* $C$*:* $\textsc{Eval}(\mathbf{SK}, i_1, \ldots, i_t, C, x) = \textsc{Comp}(T, x)$
2. ***Uniqueness****: there exists no values* $(PK, \mathbf{VK}, C, x, Z_1, Z_2)$ *where* $Z_1 = ((i_1, z_{i_1}), \ldots, (i_t, z_{i_t}))$ *and* $Z_2 = ((j_1, z_{j_1}), \ldots, (j_t, z_{j_t}))$. *s.t.*
   (a) $\textsc{Gen-Vfy}(PK, \mathbf{VK}) = 1$
   (b) *for* $k = 1, \ldots, t$:
      - $\textsc{Shr-Vfy}(PK, vk_{i_k}, z_{i_k}, x) = 1$.
      - $\textsc{Shr-Vfy}(PK, vk_{j_k}, z_{j_k}, x) = 1$.
   (c) *Let* $\mathbf{Z}_i = (z_{i_1}, \ldots, z_{i_t})$ *and* $\mathbf{Z}_j = (z_{j_1}, \ldots, z_{j_t})$. *Then,*

$$\textsc{Combine}(PK, C, \mathbf{Z}_i, x) \neq \textsc{Combine}(PK, C, \mathbf{Z}_j, x)$$

[11] Without loss of generality, we will always assume that all the $t$ partial evaluations $z'_i$ satisfy $\textsc{Shr-Vfy}(PK, vk_i, z'_i) = 1$ (else, we output $\perp$ before calling $\textsc{Combine}$). See also the definition of $\textsc{Eval}$ below to explicitly model this assumption.

3. ***Pseudorandomness under D-Core (\$-DCore)**: for any PPT algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}$ does not make query $(j, C, x)$ to* $\textsc{Prove}(\mathbf{SK}, \cdot, \cdot, \cdot)$*, for $j \notin \{i_1, \ldots, i_{t-1}\}$ for variables $i_1, \ldots, i_{t-1}, \mathbf{SK}, C, x$ defined below,*

*where $\mathbf{SK}' = (sk_{i_1}, \ldots, sk_{i_{t-1}})$.*

We present a construction of our threshold EVRF in section 5.2.

**Remark 10.** *For simplicity, in the above definition, we assume honest key generation and do not explicitly address distributed key generation. Even with this simplification, the existence of the* Gen-Vfy *algorithm ensures the users of the system that the public key $(PK, \mathbf{VK})$ is "consistent "and was generated properly. Moreover, our construction, given in Section 5.2, can easily achieve efficient distributed key generation using techniques of Gennaro et al. [29].*

**Remark 11.** *Note that when $t = n = 1$, our threshold EVRF implies the the standard EVRF definition (Definition 5), where* Post *algorithm first runs* Shr-Vfy *on the single share $z$ and then, if successful, runs* Combine *to produce the final output $y$. For $n > 1$, however, we find it extremely convenient that we can separately check the validity of each share, and be guaranteed to compute the correct output the moment $t$ servers return consistent (i.e.,* Shr-Vfy*'ed) shares $z_i$.*

## 5.2 Construction of Threshold (or Distributed) EVRFs

Our non-interactive threshold EVRF is given in Construction 3. It combines elements of our standard EVRF from Construction 2 with the ideas of Shamir's Secret Sharing [46], Feldman VSS [26], and the fact that the correctness of all computations is easily verified using the pairing.

**Security Analysis.** To check **Distribution-Correctness**, we observe that $A = g^a$, $S = g^{ar}$, and $R = g^r$. Therefore, $\textsc{Comp}(T = (R, S), x) = e(H(R, x), S) = e(H(R, x), g)^{ar}$. By definition, we have that:

$$\textsc{Eval}(PK, \mathbf{SK}, i_1, \ldots, i_t, R, x) = e(\prod_{j=1}^{t} z_{i_j}^{\lambda_j}, R)$$

$$e(\prod_{j=1}^{t} z_{i_j}^{\lambda_j}, R) = e(\prod_{j=1}^{t} H(R, x)^{a_{i_j} \cdot \lambda_j}, R) = e(H(R, x)^{\sum_{j=1}^{t} a_{i_j} \cdot \lambda_j}, R)$$

However, we know that $a = \sum_{j=1}^{t} a_{i_j} \cdot \lambda_j$. Therefore,

$$e(H(R, x)^{\sum_{j=1}^{t} a_{i_j} \cdot \lambda_j}, R) = e(H(R, x)^a, g^r) = e(H(R, x), g)^{ar}$$

To check **Uniqueness**, we are given: $(PK, \mathbf{VK} = (vk_1, \ldots, vk_n), R, x, Z_1, Z_2)$ where

$$Z_1 = ((i_1, z_{i_1}), \ldots, (i_t, z_{i_t})); \; Z_2 = ((j_1, z_{j_1}), \ldots, (j_t, z_{j_t})).$$

– $\textsc{Gen-Vfy}(PK, \mathbf{VK}) = 1$ implies that $a_0, a_1, \ldots, a_n$ where $g^{a_0} = PK$ and $g^{a_i} = vk_i$ all lie on a consistent polynomial $f$ of degree $t - 1$. Thus, there exist $\lambda_1, \ldots, \lambda_t \in \mathbb{Z}_p$ such that

**Protocol** Non-Interactive Threshold EVRF

GEN($1^k$)

Sample a random $(t-1)$ degree polynomial $f \in \mathbb{Z}_p^*[X]$.
Compute $a = f(0)$, $A_0 = g^a$.
**for** $i = 1, \ldots, n$ **do**
  Compute $a_i = f(i), A_i = g^{a_i}$.
**return** $PK = (g, A_0)$, $\mathbf{SK} = (a_1, \ldots, a_n)$, $\mathbf{VK} = (A_1, \ldots, A_n)$, with server $i$ getting secret key $sk_i = a_i$ and verification key $vk_i = A_i$.

GEN-VFY($PK, \mathbf{VK}$)

Parse $PK = (g, A_0), \mathbf{VK} =, (A_1, \ldots, A_n))$.
**for** $i = t, \ldots, n$ **do**
  Compute Lagrange coefficients $\lambda_{i,0} \ldots, \lambda_{i,t-1}$ s.t. $f(i) = \sum_{j=0}^{t-1} \lambda_{i,j} \cdot f(j)$.
  Each $\lambda_{i,j}$ is a fixed constant.
  **if** $A_i \neq \prod_{j=0}^{t-1} A_j^{\lambda_{i,j}}$ **then**
    **return** 0
**return** 1

ENCAP($PK$)

Parse $PK = (g, A_0)$.
Sample $r \in_r \mathbb{Z}_p^*$.
Compute $R = g^r, S = A_0^r$.
**return** ciphertext $C = R$ and trapdoor $T = (R, S)$.

COMP($T, x$)

Parse $T = (R, S)$.
Compute $y = e(H(R, x), S)$.
**return** $y$.

SPLIT($PK, C'$)

Parse $PK = (g, A_0)$, $C' = R'$.
**return** $R'_1 = R', \ldots, R'_n = R'$.

D-CORE($SK_i, R'_i, x$)

Parse $SK_i = a_i, R'_i = R'$.
Compute partial output $z_i = H(R'_i, x)^{a_i}$.
**return** $z_i$.

SHR-VFY($PK, VK_i, z'_i, x$)

Parse $PK = (g, A_0), VK_i = A_i$.
**if** $e(z'_i, g) \neq e(H(R'_i, x), A_i)$ **then**
  **return** $\perp$.

COMBINE($PK, C', z'_{i_1}, \ldots, z'_{i_t}, x$)

Parse $PK = (g, A_0)$, $C' = R'$.
Compute Lagrange coefficients $\lambda_1 \ldots, \lambda_t$ s.t. $f(0) = \sum_{j=1}^{t} \lambda_j \cdot f(i_j)$.
Note that these $\lambda_j$'s only depend on indices $i_1, \ldots, i_t$.
Compute $z' = \prod_{j=1}^{t} (z'_{i_j})^{\lambda_j}$.
**return** $y = e(z', R')$.

Construction 3: TEVRF = (GEN, GEN-VFY, ENCAP, COMP, SPLIT, D-CORE, SHR-VFY, COMBINE).

$f(0) = \sum_{\ell=1}^{t} \lambda_\ell \cdot f(i_\ell)$ and $\lambda'_1, \ldots, \lambda'_t \in \mathbb{Z}_p$ such that $f(0) = \sum_{\ell=1}^{t} \lambda_\ell \cdot f(j_\ell)$. Therefore, we have that:

$$A = \prod_{\ell=1}^{t} {vk_{i_\ell}}^{\lambda_\ell} = \prod_{\ell=1}^{t} {vk_{j_\ell}}^{\lambda'_\ell} \tag{2}$$

– We also know that for $\ell = 1, \ldots, t$, SHR-VFY$(PK, vk_{i_\ell}, z_{i_\ell}, x) = 1$ and SHR-VFY$(PK, vk_{j_\ell}, z_{j_\ell}, x) = 1$. Therefore, we have that for $\ell = 1, \ldots, t$:

$$e(z_{i_\ell}, g) = e(H(R, x), vk_{i_\ell});\ e(z_{j_\ell}, g) = e(H(R, x), vk_{j_\ell}) \tag{3}$$

– We will now show that the 2 outputs of COMBINE must be equal. Here we we will write $R = g^r$ for some $r$,

$$\text{COMBINE}(PK, R, z_{i_1}, \ldots, z_{i_t}, x) = e(\prod_{\ell=1}^{t} z_{i_\ell}^{\lambda_\ell}, R) = \prod_{\ell=1}^{t} e(z_{i_\ell}, g)^{r \cdot \lambda_\ell}$$

From Equation (3):

$$\prod_{\ell=1}^{t} e(z_{i_\ell}, g)^{r \cdot \lambda_\ell} = \prod_{\ell=1}^{t} e(H(R, x), vk_{i_\ell})^{r \cdot \lambda_\ell} = e(H(R, x), \prod_{\ell=1}^{t} vk_{i_\ell}^{\lambda_\ell})^r$$

From Equation (2), we have that:

$$e(H(R,x), \prod_{\ell=1}^{t} vk_{i_\ell}^{\lambda_\ell})^r = e(H(R,x), \prod_{\ell=1}^{t} vk_{j_\ell}^{\lambda'_\ell})^r = \prod_{\ell=1}^{t} e(H(R,x), vk_{j_\ell})^{r \cdot \lambda'_\ell}$$

We again use Equation (3) to conclude the proof. Finally, we prove the following result in Section C.3.

**Theorem 9.** *If Construction 2 satisfies the* ***$-Core*** *property of standard* EVRF*, then Construction 3 satisfies the* ***$-DCore*** *property of threshold* EVRF*. By Theorem 7, it follows that Construction 3 satisfies the* ***$-DCore*** *property under the* BDDH *assumption in the random oracle model.*

# 6 Delegatable Encapsulated Verifiable Random Functions

In this section, we formally introduce the primitive known as a Delegatable EVRF in Section 6.1. This definition captures different levels of delegatability and we present constructions that satisfy these levels in Sections 6.2,6.3, and 6.4. The security proofs are deferred to the appendix.

## 6.1 Definition of Delegatable EVRFs

In this work, we will be interested in a stronger type of delegatable EVRFs where anybody can check if two ciphertexts $C_1$ and $C_2$ "came from the same place". This is governed by the "comparison" procedure $\textsc{Same}(PK_1, C_1, PK_2, C_2)$ which outputs 1 only if Equation (1) holds. This procedure will have several uses. First, it allows the owner of $SK_2$ to be sure that the resulting ciphertext $C_2$ indeed encapsulates the same VRF under $PK_2$ as $C_1$ does under $PK_1$. Second, it will allow us to cleanly define a "trivial" attack on the pseudorandomness of delegatable EVRFs. See also Remark 13.

Before we define the syntax and the security of delegatable EVRFs, we include a brief exposition on the nuances in the syntax and security of such a primitive.

Secretly-Delegatable vs Publicly-Delegatable EVRFs. It is fairly obvious from the security of EVRFs that the delegation procedure must use the secret key $SK_1$ of the delegating party. The big distinction/subtlety comes from whether or not such delegation also requires the secret key $SK_2$ of the receiving party. In our basic notion, defined below, we will allow such a dependence. However, for completeness, we also define *publicly-delegatable* EVRFs, where only the public key $PK_2$ is needed. A priori, publicly delegatable EVRFs have the advantage that the delegation does not need the cooperation of the receiving party. However, all our secretly-delegatable schemes will have a trivial implementation, where the sender can use $SK_1$ to non-interactively convert $C_1$ into the "$C_1$-specific" trapdoor $T_1$ of the EVRF, which it can (securely) send to the receiving party. In turn, the receiving party can use the secret key $SK_2$ to convert $T_1$ to the corresponding delegated ciphertext $C_2$. Thus, all our concrete secret-key delegatable EVRFs will have no "usability disadvantages" compared to publicly-delegatable EVRFs.

Bounded Delegation. Additionally, the definition we present is for unbounded delegation, where any delegated ciphertext can be further delegated. We could also restrict the definition to *$t$-delegatable*, with $t = 1$ being an important special case, where **Delegation-Completeness** is only required to hold for up to $t$ iterated delegations starting from any ciphertext $C_1$ output by the encapsulation procedure Encap. We will consider such a variant in Section 6.4.

Pseudorandomness of Delegatable EVRFs. To capture the pseudorandomness property of delegatable EVRFs, it is clear that our definition should give the attacker the ability to call the delegation oracle Del. However, there are several subtleties in such a definition which we list below:

- Should we allow delegation queries from target $SK_1$ only to honestly generated keys $(PK_2, SK_2)$ (for which the attacker does not know $SK_2$), or shall we allow the attacker $\mathcal{A}$ to specify such keys adversarially?

- For secretly-delegatable schemes, should the attacker $\mathcal{A}$ only have access to "OUT" oracle $\text{Del}(SK_1, \cdot, \cdot)$, or should we also give $\mathcal{A}$ the "IN" oracle $\text{Del}(\cdot, \cdot, SK_1)$ as well?[12] This corresponds to the attacker $\mathcal{A}$ tricking the user $U$ to get some malicious EVRF from $\mathcal{A}$, only to force $U$ to use its secret key $SK_1$ in a way that will help the attacker break some honest EVRF owned by $U$.

- What is the definitional security effect of studying one-time vs $t$-time vs unbounded-time delegatable schemes? As we will see, the security definitions *will not change syntactically* when restricted to $t$-time delegatable schemes. In particular, we will not explicitly limit the number of times the attacker can attempt to iteratively call the delegation oracle. However, since in such schemes the correctness is no longer required when delegating more than $t$ times, it will be easier to make $t$-delegatable schemes secure when $t$ is smaller.

- How to prevent the attacker from "trivial" attacks, where one can delegate ciphertext $C_1$ under $PK_1$ to $C_2$ under $PK_2$, and then "break" $C_1$ by asking an evaluation query on $C_2$? This is where the comparison procedure Same will be handy, as it allows us to precisely exclude all such ciphertexts $C_2$ which "originated" from $C_1$, while still allowing the attacker to try all other ciphertexts.[13]

Now, we can define the oracles. To adequately capture the discussed nuances, we define the following oracles to the attacker:

1. $\text{Reg}(1^k)$: registration oracle. It maintains a global variable $q$, initially 0, counting the number of non-compromised users. A call to Reg: (a) increments $q$; (b) calls $(PK_q, SK_q) \leftarrow \text{Gen}(1^k)$, (c) records this tuple $(q, PK_q, SK_q)$ in a global table not accessible to the attacker; (d) returns $(q, PK_q)$ to the attacker.

2. $\text{HProve}(i, C, x)$: honest evaluation oracle. Here $1 \le i \le q$ is an index, $C$ is a ciphertext, and $x$ in an input. The oracle returns $\text{Prove}(SK_i, C, x) = \text{Core}(SK_i, \text{Split}(PK_i, C), x)$.

3. $\text{HDel}(i, C, j)$: honest delegation oracle. Here $1 \le i, j \le q$ are two indices, and $C$ is a ciphertext. The oracle returns $C' = \text{Del}(SK_i, C, SK_j)$ (or $\text{Del}(SK_i, C, PK_j)$ in the publicly-delegatable case).

4. $\text{OutDel}(i, C, SK/PK)$: "Out" delegation oracle. Here $1 \le i \le q$ is an index, $C$ is a ciphertext, and $PK$ or $SK$ (depending on whether scheme is publicly-delegatable or not) is any public/secret key chosen by the attacker. The oracle returns $C' = \text{Del}(SK_i, C, SK/PK)$.

[12] Clearly, this is a moot issue for publicly-delegatable schemes.

[13] In Definition 21 we will give an even stronger (and optimal) legality condition; some of our schemes will satisfy even this stronger notion.

5. $\text{InDel}(SK, C, i)$: "In" delegation oracle. Here $1 \leq i \leq q$ is an index, and $C$ is a ciphertext, and $SK$ is any secret key chosen by the attacker. The oracle returns $C' = \text{Del}(SK, C, SK_i)$. Notice, this oracle is interesting only in the secretly-delegatable case.

Consequently, we can define three levels of pseudorandomness security for delegatable EVRFs.

**Definition 10.** *An EVRF* $= (\text{Gen}, \text{Encap}, \text{Comp}, \text{Split}, \text{Core}, \text{Post})$ *is* delegatable *if there exists polynomial-time procedures* Del *and* Same*, such that:*

- $\text{Del}(SK_1, C_1, SK_2) = C_2$ *for the (default)* secretly-delegatable *variant;*
- $\text{Del}(SK_1, C_1, PK_2) = C_2$ *for the* publicly-delegatable *variant.*
- $\text{Same}(PK_1, C_1, PK_2, C_2) = \beta \in \{0, 1\}$.

*Before we define the security properties, it is useful to define the following shorthand functions:*

- $\text{Prove}(SK_i, C, x) = \text{Core}(SK_i, \text{Split}(PK_i, C), x)$
- $\text{Eval}(SK, C, x) = \text{Post}(PK, \text{Prove}(SK, C, x), C, x)$

*In addition to the standard* EVRF *properties of* ***Evaluation-Correctness*** *and* ***Uniqueness*** *, we require the following security properties from a delegatable* EVRF*:*

1. ***Delegation-Completeness****: for any valid* $(PK_1, SK_1)$, $(PK_2, SK_2)$*, and ciphertext* $C_1$*,*

$$\text{Del}(SK_1, C_1, SK_2/PK_2) = C_2 \implies \text{Same}(PK_1, C_1, PK_2, C_2) = 1$$

2. ***Delegation-Soundness****: for any valid* $(PK_1, SK_1), (PK_2, SK_2)$*, and ciphertexts* $C_1$, $C_2$

$$\begin{aligned} &\text{Same}(PK_1, C_1, PK_2, C_2) = 1 \implies \\ &\qquad \forall x \ \text{Eval}(SK_1, C_1, x) = \text{Eval}(SK_2, C_2, x) \end{aligned}$$

   *Moreover, if we have* $PK_1 = PK_2$*, then* $C_1 = C_2$.

1. ***Pseudorandomness under Core (\$-Core)****: for any legal PPT attacker* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, where legality of* $\mathcal{A}$ *and appropriate delegation oracle(s)* $\mathcal{O}$ *are defined separately for each notion:*

$$\Pr\left[\, b = b' \;\middle|\; \begin{array}{c} (1, PK_1) \leftarrow \text{Reg}(1^k); \\ (C_1, T_1) \leftarrow \text{Encap}(PK_1); \\ (x, st) \leftarrow \mathcal{A}_1^{\text{Reg,HProve},\mathcal{O}}(PK_1, C_1); \\ y_0 = \text{Comp}(T_1, x); \quad y_1 \leftarrow \{0,1\}^{|y_0|}; \\ b \leftarrow \{0,1\}; \quad b' \leftarrow \mathcal{A}_2^{\text{Reg,HProve},\mathcal{O}}(y_b, st) \end{array} \right] \leq \frac{1}{2} + \mathit{negl}(k)$$

   (a) ***Basic-\$-Core****: $\mathcal{A}$ has 1 delegation oracle* $\mathcal{O} = \text{HDel}$.
   *Legality of* $\mathcal{A}$*: no call to* $\text{HProve}(i, C', x)$ *s.t.*
   $\text{Same}(PK_1, C_1, PK_i, C') = 1$.

   (b) ***Uni-\$-Core****: $\mathcal{A}$ has 2 delegation oracles* $\mathcal{O} = (\text{HDel}, \text{OutDel})$.
   *Legality of* $\mathcal{A}$*: no call to* $\text{HProve}(i, C', x)$ *or*
   $\text{OutDel}(i, C', *)$ *s.t.* $\text{Same}(PK_1, C_1, PK_i, C') = 1$.

(c) ***Bi-$-Core**: $\mathcal{A}$ has 3 delegation oracles $\mathcal{O} = (\text{HDEL}, \text{OUTDEL}, \text{INDEL})$.*
*Legality of $\mathcal{A}$: same as that of **Uni-$-Core**.*

**Remark 12.** ***Delegation-Completeness** and **Delegation-Soundness** easily imply **Delegation-Correctness** which was advocated in Equation (1):*

$$\text{DEL}(SK_1, C_1, SK_2/PK_2) = C_2 \implies \forall x \ \text{EVAL}(SK_1, C_1, x) = \text{EVAL}(SK_2, C_2, x)$$

**Remark 13.** *The legality condition on the attacker is necessary, as evaluating EVRF on the "same" ciphertext $C'$ as the challenge ciphertext $C_1$ breaks pseudorandomness (by delegation-soundness). However, it leaves open the possibility for the attacker to find such equivalent ciphertext $C'$* without building some explicit "delegation path" *from the challenge ciphertext $C_1$. Indeed, in Definition 21 we will give an even stronger legality condition on $\mathcal{A}$, and some (but not all) of our schemes will meet it. For applications, however, we do not envision this slight definitional gap to make any difference. Namely, the higher-level application will anyway need some mechanism to disallow any "trivial" attacks. We expect this mechanism will explicitly use our* SAME *procedure, rather than keep track of the tree of "delegation paths" originating from $C_1$, which could quickly become unmanageable.*

**Remark 14.** *It is easy to observe the following implications:*

$$\textbf{Bi-\$-Core} \implies \textbf{Uni-\$-Core} \implies \textbf{Basic-\$-Core} \implies \textbf{\$-Core}$$

Here, the last implication uses the fact that $C_1$ is the only ciphertext equivalent to $C_1$ under $PK_1$. Thus, bidirectional delegation security is the strongest of all the notions.

**Remark 15.** *One could also consider EVRFs which are simultaneously threshold and delegatable. In this case, $n_1$ servers for the sender's EVRFs will communicate with $n_2$ servers for the receiver's EVRF to help convert a ciphertext $C_1$ for the sender EVRF into a corresponding ciphertext $C_2$ for the receiver EVRF. We leave this extension to future work.*

## 6.2 Construction of Basic Delegatable EVRF

We now show that our original EVRF Construction 2 can be extended to make it basic-delegatable. The idea is to separate the role of the "handle" $R$ hashed under $H$ inside the CORE procedure from the one used in the preprocessing. For technical reasons explained below, we will also hash the public key $A$ when evaluating the EVRF. The construction is presented as Construction 4.

OBSERVATIONS. We notice that, since $R = D$ initially, the resulting EVRF before the delegation is the same as the one we defined in Section 4.4, except (a) we also include the public key $A$ under the hash $H$ during both ENCAP and CORE; and (b) we perform the delegation check $e(A', R') \stackrel{?}{=} e(A, D')$ in the split procedure SPLIT, which is trivially true initially, as $A' = A$ and $R' = D' = R$. Thus, **Evaluation-Correctness** trivially holds, as before. For the same reason, **Uniqueness** trivially holds as well.

The importance of change (a) comes from the fact that challenge ciphertext $C = (A, R, D)$ no longer includes only the value $R$, even though the value $R$ would be all that is needed to actually evaluate our EVRF, had we not included $A$ under the hash $H$. In particular, the attacker $\mathcal{A}$ given challenge $C = (A, R, R)$, can easily produce $C' \neq C$ by setting $C' = (A^2, R, R^2)$. $C'$ passes the

**Protocol** Basic Delegatable EVRF

GEN($1^k$)
Sample $a \in_r \mathbb{Z}_p^*$
Compute $A = g^a \in \mathbb{G}$.
**return** $SK = a$ and $PK = (g, A)$.

ENCAP($PK$)
Parse $PK = (g, A)$.
Sample $r \in_r \mathbb{Z}_p^*$.
Compute $R = D = g^r, S = A^r$.
**return** ciphertext $C = (A, R, D)$ and trapdoor $T = (A, R, S)$.

COMP($T, x$)
Parse $T = (A, R, S)$.
Compute $y = e(H(A, R, x), S)$.
**return** $y$.

DEL($SK_1, C_1, SK_2$)
Parse $SK_1 = a_1, SK_2 = a_2, C_1 = (A, R, D_1)$.
**if** $e(A, R) \neq e(g^{a_1}, D_1)$ **then**
  **return** $\perp$.
**else**
  Compute $D_2 = D_1^{a_1/a_2}$ where $a_1/a_2 = a_1 \cdot (a_2)^{-1} \mod p$.
  **return** $C_2 = (A, R, D_2)$.

SPLIT($PK, C'$)
Parse $PK = (g, A)$, $C' = (A, R', D')$.
**if** $e(A', R') \neq e(A, D')$ **then**
  **return** $\perp$.
**else**
  **return** $(A', R')$.

CORE($SK, C', x$)
Parse $SK = a, C' = (A', R', D')$.
Compute partial output $z = H(A', R', x)^a$.
**return** $z$.

POST($PK, z', C', x$)
Parse $PK = (g, A)$, $C' = (A', R', D')$
**if** $e(z', g) \neq e(H(A', R', x), A)$ **then**
  **return** $\perp$.
**else**
  Compute full output $y' = e(z', D')$.
  **return** $y'$.

SAME($PK_1, C_1, PK_2, C_2$)
Parse $PK_1 = (g, A_1), PK_2 = (g, A_2), C_1 = (A, R, D_1), C_2 = (A', R', D_2)$.
**if** $(A, R) \neq (A', R')$ or $e(A_1, D_1) \neq e(A_2, D_2)$ **then**
  **return** $\perp$.

Construction 4: Basic Delegatable $\text{DEVRF}_1$ = (GEN, ENCAP, COMP, SPLIT, CORE, POST, DEL, SAME).

delegation check $e(A^2, R) = e(A, R^2)$, but clearly produces the same partial output $z = H(R, x)^a$ as the challenge ciphertext, trivially breaking the **$-Core** property. Instead, by also hashing the public key, the oracle call PROVE($C', x$) would return $z' = H(A^2, R, x)^a$, which is now unrelated to $z = H(A, R, x)^a$, foiling the trivial attack.

The importance of change (b) comes from ensuring that a valid ciphertext $(A', R', D')$ determines the value $D'$ *information-theoretically* from the values $(A', R')$ (and the public key $A$), because the condition $e(A', R') = e(A, D')$ uniquely determines $D'$. Thus, it is OK that the CORE procedure only passes the values $(A', R')$ under the random oracle $H$.

DELEGATION. To check **Delegation-Completeness**, notice that valid delegation of $(A, R, D_1)$ outputs $(A', R', D_2)$, where $(A', R') = (A, R)$ and $D_2 = D_1^{a_1/a_2}$, which implies that

$$e(A_2, D_2) = e(g^{a_2}, D_1^{a_1/a_2}) = e(g^{a_1}, D_1) = e(A_1, D_1)$$

which means $\text{SAME}(A_1, (A, R, D_1), A_2, (A', R', D_2)) = 1$ indeed.

For **Delegation-Soundness**, given $C_1 = (A, R, D_1)$ and $C_2 = (A', R', D_2)$ satisfying $(A', R') = (A, R)$ and $e(A_1, D_1) = e(A_2, D_2)$, we can see that the delegation checks $e(A, R) \stackrel{?}{=} e(A_1, D_1)$ and $e(A', R') \stackrel{?}{=} e(A_2, D_2)$ are either both false or true simultaneously. Moreover, by writing $A_1 = A_2^{a_1/a_2}$, the second equation implies that $D_2 = D_1^{a_1/a_2}$. In particular, if $A_1 = A_2$, we have $C_1 = C_2$; and, in general, when $(A', R') = (A, R)$ and $D_2 = D_1^{a_1/a_2}$, for any $x$, we know: $\text{EVAL}(a_2, (A, R, D_2), x) = e(H(A, R, x)^{a_2}, D_2)$.

However, that can be rewritten as

$$e(H(A, R, x)^{a_2}, D_1^{a_1/a_2}) = e(H(A, R, x)^{a_2}, D_1^{a_1/a_2}) = e(H(A, R, x)^{a_1}, D_1)$$

which concludes the proof.

We reiterate that though our delegation is secretly-delegatable, as $D_2$ depends on $a_2$, in practice the owner Alice of $a_1$ will simply send the trapdoor value $T_1 = D_1^{a_1}$ to the owner Bob of $a_2$ over secure channel (say, encrypted under a separate public key), and Bob can then compute $D_2 = T_1^{1/a_2}$. In particular, this does not leak any extra information beyond $(D_2, a_2)$ to Bob, as $T_1 = D_2^{a_2}$ is efficiently computable from $D_2$ and $a_2$. Also, the delegation check does not require any of the secret keys. Despite that, it ensures that only properly delegated ciphertexts can be securely re-delegated again. We will critically use it in Section C.4 to prove the following:

**Theorem 11.** *The basic delegatable EVRF, given in Construction 4, satisfies the* ***Basic-$-Core*** *property under the* BDDH *assumption in the random oracle model.*

Delegation Attack on Stronger Legality. We briefly mentioned in Section 6.1 that one could require a stronger legality condition to say that the only way to distinguish the evaluation of $C$ on $x$ from random is to honestly delegate $C$ to some honest user (possibly iteratively), getting ciphertext $C'$, and then ask this user to evaluate EVRF on $x$.

Here we show that our construction does not satisfy this notion. Consider challenge ciphertext $C_1 = (A_1, R_1, R_1)$ under public key $A_1$. Construct $C_1' = (A_1, R_1^2, R_1^2)$. $C_1'$ will satisfy the delegation check, so we could ask to delegate $C'$ to public key $A_2$. We get $C_2' = (A_1, R_1^2, (R_1^2)^{a_1/a_2}) = (A_1, R_1^2, (R_1^{a_1/a_2})^2)$. By taking square roots from the last two components, we get $C_2 = (A_1, R_1, R_1^{a_1/a_2})$. Notice, $\textsc{Same}(A_1, C_1, A_2, C_2) = 1$ is true, so our original definition does *not* permit the attacker to evaluate $\textsc{HProve}(2, C_2, x)$ (which clearly breaks the scheme). However, since we obtained $C_2$ *without* asking the delegate $C_1$ itself (instead, we asked a different ciphertext $C_1'$), the stronger notion would have allowed the attacker to call $\textsc{HProve}(2, C_2, x)$ and break the scheme.

### 6.3 Construction of Uni- and Bidirectional Delegatable EVRF

Next, we extend the construction from the previous EVRF construction to also handle delegation *to* (and, under a stronger assumption, *from*) potentially untrusted parties. The idea is to add a "BLS signature" [11] $\sigma$ in the Encap procedure which will prove that the initial ciphertext was "well-formed". This makes it hard for the attacker to maul a valid initial ciphertext $C$ into a related ciphertext $C'$, whose delegation might compromise the security of $C$. The public verifiability of the signature $\sigma$ will also make it easy to add a "signature check" to the "delegation check" we already used in our scheme, to ensure that the appropriate pseudorandomness property is not compromised. This is presented as Construction 5.

**Security Analysis.** Since $\text{DEVRF}_2$ is essentially the same as $\text{DEVRF}_1$, its correctness follows the same argument. In particular, we notice that the original signature $\sigma$ indeed satisfies our signature check:

$$e(H'(A,R),R) = e(H'(A,R),g^r) = e(H'(A,R)^r, g) = e(\sigma, g)$$

Similar to the delegation check, the signature check, $e(H'(A',R'),R') \stackrel{?}{=} e(\sigma', g)$, is important to ensure that the value $\sigma'$ is information-theoretically determined from the value $(A', R')$, so it is fine to not include $\sigma$ under $H$.

Also, since the delegation procedure Del simply copies the values $A, R$ and $\sigma$, and only modifies the value $D_1$, the **Delegation-Completeness** and **Delegation-Soundness** of $\text{DEVRF}_2$ holds

**Protocol** Delegatable EVRF

$\underline{\textsc{Gen}(1^k)}$

Sample $a \in_r \mathbb{Z}_p^*$.
Compute $A = g^a \in \mathbb{G}$.
**return** $SK = a$ and $PK = (g, A)$.

$\underline{\textsc{Encap}(PK)}$

Parse $PK = (g, A)$.
Sample $r \in_r \mathbb{Z}_p^*$.
Compute $R = D = g^r, S = A^r, \sigma = H'(A, R)^r$.
**return** ciphertext $C = (A, R, D, \sigma)$ and trapdoor $T = (A, R, S)$.

$\underline{\textsc{Comp}(T, x)}$

Parse $T = (A, R, S)$.
Compute $y = e(H(A, R, x), S)$.
**return** $y$.

$\underline{\textsc{Del}(SK_1, C_1, SK_2)}$

Parse $SK_1 = a_1, SK_2 = a_2, C_1 = (A, R, D_1, \sigma)$.
**if** $e(A, R) \neq e(g^{a_1}, D_1)$ or $e(H'(A, R), R) \neq e(\sigma, g)$ **then**
  **return** $\perp$.
**else**
  Compute $D_2 = D_1^{a_1/a_2}$ where $a_1/a_2 = a_1 \cdot (a_2)^{-1} \mod p$.
  **return** $C_2 = (A, R, D_2)$.

$\underline{\textsc{Split}(PK, C')}$

Parse $PK = (g, A)$, $C' = (A', R', D', \sigma')$.
**if** $e(A', R') \neq e(A, D')$ or $e(H'(A', R'), R') \neq e(\sigma', g)$ **then**
  **return** $\perp$.
**else**
  **return** $(A', R')$.

$\underline{\textsc{Core}(SK, C', x)}$

Parse $SK = a, C' = (A', R', D', \sigma')$.
Compute partial output $z = H(A', R', x)^a$.
**return** $z$.

$\underline{\textsc{Post}(PK, z', C', x)}$

Parse $PK = (g, A)$, $C' = (A', R', D', \sigma')$
**if** $e(z', g) \neq e(H(A', R', x), A)$ **then**
  **return** $\perp$.
**else**
  Compute full output $y' = e(z', D')$.
  **return** $y'$.

$\underline{\textsc{Same}(PK_1, C_1, PK_2, C_2)}$

Parse $PK_1 = (g, A_1), PK_2 = (g, A_2), C_1 = (A, R, D_1, \sigma), C_2 = (A', R', D_2, \sigma')$.
**if** $(A, R, \sigma) \neq (A', R', \sigma')$ or $e(A_1, D_1) \neq e(A_2, D_2)$ **then**
  **return** $\perp$.

Construction 5: $\text{DEVRF}_2 = (\textsc{Gen}, \textsc{Encap}, \textsc{Comp}, \textsc{Split}, \textsc{Core}, \textsc{Post}, \textsc{Del}, \textsc{Same})$.

as it did for $\text{DEVRF}_1$, since the signature check is not affected by changing $D_1$ to $D_2 = D_1^{a_1/a_2}$. In particular, similar to the delegation checks, both signature checks are either simultaneously true or false.

More importantly, in Section C.5 we also show how the addition of the "BLS signature" $\sigma$ and the new signature check allow us to prove the following theorem:

**Theorem 12.** *The delegatable EVRF given in Construction 5 satisfies the* ***Uni-$-Core*** *property under the* BDDH *assumption in the random oracle model.*

Finally, we also show that the same construction also satisfies the strongest *bidirectional-delegation* security, but now under a much stronger iBDDH assumption. In fact, for this result, we will even show a *stronger legality* condition mentioned earlier: the only way to break $\text{DEVRF}_2$ is to trivially delegate it "out" to the attacker, or delegate it to the honest user, and then ask the user to evaluate on challenge $x$. We define this formally in Section C.6 (see Definition 21), where we also show the following result:

**Theorem 13.** *The delegatable EVRF given in Construction 5 satisfies the* ***Bi-$-Core*** *property under the interactive* iBDDH *assumption in the random oracle model. It satisfies the strongest possible legality condition for the attacker (see Definition 21).*

## 6.4 Construction of One-time Delegatable EVRF

Note that the bidirectional-delegation security of Construction 5 relied on a very strong inversion-oracle BDDH (iBDDH) assumption, which is interactive and not well studied. For applications where we only guarantee security after a single delegation, we could prove bidirectional-delegation under a much reasonable extended BDDH (eBDDH) assumption. More precisely, any party $P$ is "safe" to do any number of "out-delegations" to other, potentially untrusted parties $P'$, but should

only accept "in-delegation" from such an untrusted $P'$ only if the delegated ciphertext $C'$ was created directly for $P'$ (and not delegated to $P'$ from somewhere else).

More formally, the one-time delegation scheme we present here is identical to the unidirectional-delegation scheme from the previous section, except we replace the "delegation check"

$$e(A, R) \stackrel{?}{=} e(A_1, D_1)$$

by a stricter "equality check":

$$(A, R) \stackrel{?}{=} (A_1, D_1)$$

which means that the ciphertext $C_1$ was directly created for public key $A_1 = A$. We call the resulting 1-time-delegatable construction $\text{DEVRF}_3$. In Section C.7 we show that $\text{DEVRF}_3$ satisfies bidirectional-delegation security, but now under a much weaker (non-interactive) eBDDH assumption:

**Theorem 14.** *The* one-time *delegatable* $\text{DEVRF}_3$ *above satisfies the* ***Bi-$-Core*** *property under the* eBDDH *assumption in the random oracle model. It satisfies the strongest possible legality condition for the attacker (see Definition 21).*

We stress that our 1-time delegatable scheme could in principle be delegated further, if the stricter delegation check $(A, R) \stackrel{?}{=} (A_1, D_1)$ is replaced by the original check $e(A, R) \stackrel{?}{=} e(A_1, D_1)$. However, by doing so the party receiving the EVRF from some untrusted source must rely on the stronger iBDDH complexity assumption.

# 7 Conclusion and Final Thoughts

In this work we introduce the idea of an encapsulated search index (ESI) that offers support for public-indexing and where the search takes sub-linear time. We also presented a generic construction of ESI from another primitive known as encapsulated verifiable random functions (EVRF). We further detailed meaningful extensions to both ESI and EVRF with support for delegation and distribution. We presented constructions of a standard EVRF and its various extensions. Indeed, obtain the following Theorem as a corollary of Theorem 6, and by using any updatable sub-linear DS with an appropriate (delegatable and/or threshold) EVRF from the earlier sections, we get:

**Theorem 15.** *We have an updatable* ESI *(Section 3.5) which*

*(a) maintains the efficiency of the non-cryptographic* DS*;*

*(b) has non-interactive* $(t, n)$ *threshold implementation for token generation (by using* TEVRF*); and*

*(c) achieves either of the following delegation security levels in the random oracle model:*

- ***Basic CCA secure*** *under* BDDH *assumption (by using* $\text{DEVRF}_1$*)*
- ***Uni CCA secure*** *under* BDDH *assumption (by using* $\text{DEVRF}_2$*)*
- ***Bi CCA secure*** *under* iBDDH *assumption (by using* $\text{DEVRF}_2$*)*
- ***One Time CCA secure*** *under* eBDDH *assumption (by using* $\text{DEVRF}_3$*)*

Commercial Product. This theorem forms the backbone of a commercially available product that has been in the market since 2020 called Atakama [3]. It serves over two-dozen enterprise customers, with the largest having over 100 users. At a high level, the commercial application is essentially the motivating application described in the Introduction, but with a few pragmatic extensions.

The code is production quality and has been deployed without any noticeable performance degradation, even for large files. Note that a typical mobile device has the capability to compute 10,000 elliptic curve multiplications (which is needed in our partial decryption step) per second, with the help of multiple cores. This number is only expected to go up with further technological advancements such as the growth of mobile GPUs. In the search functionality, a user can enter one or several keywords. The system then sequentially searches each file using the ESI that has been built leading to a total complexity proportional to the product of the number of keywords, the number of files, and the ESI search time. By using a blinded bloom filter as the data structure, the application achieves a constant time search dictionary.[14] Currently, searching 1000 files with up to 4 keywords (or 2000 files with a maximum of 2 keywords) can be accomplished in about 2 seconds on a standard mobile phone. The application already uses the distributed token generation and the search delegation capabilities of our underlying ESI. We present additional details in Section A.

## References


[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[2] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1993–2010, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[3] Atakama. Secure your files multi-factor encryption, 2022. https://www.atakama.com/.

[4] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. In Osvaldo Gervasi, Beniamino Murgante, Antonio Laganà, David Taniar, Youngsong Mun, and Marina L. Gavrilova, editors, *Computational Science and Its Applications – ICCSA 2008*, pages 1249–1259, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[5] Steven M. Bellovin and William R. Cheswick. Privacy-enhanced searches using encrypted bloom filters. Cryptology ePrint Archive, Report 2004/022, 2004. http://eprint.iacr.org/2004/022.

[6] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.


[14] Of course, the indexing step is proportional to the size of the file.

[7] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 226–243, San Jose, CA, USA, February 13–17, 2006. Springer, Heidelberg, Germany.

[8] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[9] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[10] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows PIR queries. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 50–67, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

[11] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

[12] Xavier Boyen, Qixiang Mei, and Brent Waters. Direct chosen ciphertext security from identity-based techniques. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 2005: 12th Conference on Computer and Communications Security*, pages 320–329, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press.

[13] Ran Canetti and Shafi Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In Stern [48], pages 90–106.

[14] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 668–679, Denver, CO, USA, October 12–16, 2015. ACM Press.

[15] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *ACNS 05: 3rd International Conference on Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455, New York, NY, USA, June 7–10, 2005. Springer, Heidelberg, Germany.

[16] Cloudflare. Cloudflare Randomness Beacon docs. https://developers.cloudflare.com/randomness-beacon/.

[17] Corestar. corestario/tendermint, October 2020. original-date: 2018-12-19T13:33:15Z.

[18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[19] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany.

[20] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 79–88, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[21] DAOBet. DAOBet (ex — DAO.Casino) to Deliver On-Chain Random Beacon Based on BLS Cryptography, May 2019. https://daobet.org/blog/on-chain-random-generator/.

[22] Dawn Xiaoding Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, pages 44–55, 2000.

[23] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 1–17, Miami, FL, USA, January 6–8, 2003. Springer, Heidelberg, Germany.

[24] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431, Les Diablerets, Switzerland, January 23–26, 2005. Springer, Heidelberg, Germany.

[25] Ratna Dutta, Rana Barua, and Palash Sarkar. Pairing-based cryptographic protocols : a survey. Cryptology ePrint Archive, Report 2004/064, 2004. http://eprint.iacr.org/2004/064/.

[26] Frank A. Feldman. Fast spectral tests for measuring nonrandomness and the DES. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 243–254, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.

[27] Steven D. Galbraith. Supersingular curves in cryptography. *Lecture Notes in Computer Science*, 2248:495–513, 2001.

[28] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. Cryptology ePrint Archive, Report 2020/096, 2020. https://eprint.iacr.org/2020/096.

[29] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

[30] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216.

[31] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably preventing DNSSEC zone enumeration. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, San Diego, CA, USA, February 8–11, 2015. The Internet Society.

[32] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). Internet-Draft draft-irtf-cfrg-vrf-07, Internet Engineering Task Force, June 2020. Work in Progress.

[33] Antoine Joux and Kim Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org/2001/003/`.

[34] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012: 19th Conference on Computer and Communications Security*, pages 965–976, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[35] Keep. The Keep Random Beacon: An Implementation of a Threshold Relay, 2020. https://docs.keep.network/random-beacon/.

[36] Veronika Kuchta and Mark Manulis. Unique aggregate signatures with applications to distributed verifiable random functions. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *CANS 13: 12th International Conference on Cryptology and Network Security*, volume 8257 of *Lecture Notes in Computer Science*, pages 251–270, Paraty, Brazil, November 20–22, 2013. Springer, Heidelberg, Germany.

[37] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 597–612, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

[38] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 120–130, New York, NY, USA, October 17–19, 1999. IEEE Computer Society Press.

[39] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Stern [48], pages 327–346.

[40] Moni Naor and Eylon Yogev. Tight bounds for sliding bloom filters. *Algorithmica*, 73(4):652–672, December 2015.

[41] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal bloom filter replacement. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, page 823–829, USA, 2005. Society for Industrial and Applied Mathematics.

[42] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122 – 144, 2004.

[43] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. Improved searchable public key encryption with designated tester. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *ASIACCS 09: 4th ACM Symposium on Information, Computer and Communications Security*, pages 376–379, Sydney, Australia, March 10–12, 2009. ACM Press.

[44] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. ETHDKG: Distributed key generation with Ethereum smart contracts. Cryptology ePrint Archive, Report 2019/985, 2019. https://eprint.iacr.org/2019/985.

[45] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27:701–717, 1980.

[46] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[47] Victor Shoup. Lower bounds for discrete logarithms and related problems. *Lecture Notes in Computer Science*, 1233:256–266, 1997.

[48] Jacques Stern, editor. *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[49] Yunhong Zhou, Na Li, Yanmei Tian, Dezhi An, and Licheng Wang. Public key encryption with keyword search in cloud: A survey. *Entropy*, 22(4), 2020.

# A ESI, in Practice

In this section, we look at the deployed product [3] with a focus on the minutiae of the implementation and the user interface. However, before we proceed, we look at a high-level overview of the application.

The application is designed to allow users to: upload files on a device (say, a desktop); encrypt these files; and search these files with the search operation being approved by multiple devices (say, a mobile phone and/or a laptop). The application outputs all the files that contain the keyword. The techniques presented in this paper are for purposes of supporting the search functionality: building, searching, and delegating the index. Thus, below we will do not discuss the encryption/decryption of files, which uses rather standard techniques.

We note that the indexed files are entirely known at the time of indexing. Furthermore, the output of the search operation is the actual list of all files containing the keyword, as opposed to a less useful answer on whether *some* file contains the keyword. Therefore, the issue of information leakage due to multiple files sharing keywords (which is of great concern in the PEKS/SSE setting) is moot. In other words, the "static" nature of ESI is totally fine for this application. Moreover, in terms of efficiency, searching is done sequentially over all files but does almost does not depend on the sizes of individual files, which is a great saving compared to PEKS. Similarly, the fact that indexing is done on the desktop without any involvement of the mobile "search approvers" is a huge benefit over SSE.

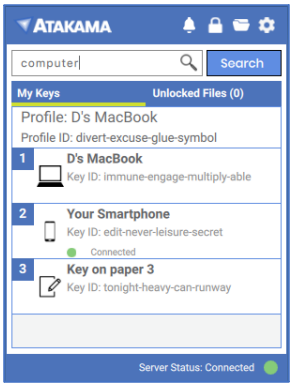
Moreover, *non-interactive* threshold implementation is extremely handy, allowing the owner of the mobile device to simply "tap approve" (see below), and not worry about losing connectivity in

between the approval process, or needing to wait for all helper devices to be online. Finally, the application supports delegating the search operation to new "desktops", which implies the need for supporting such operations without spending precious resources rebuilding the index. In short, the motivating application is to build a static index that achieves sub-linear search, with support for threshold search approvals and delegation of such approvals, which is precisely the ESI setting.

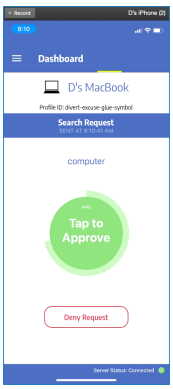
The images below contain fictitious company name and filenames to preserve anonymity.

## A.1 Indexing and Search Functionality

- The software begins by creating an index for every file it encrypts. This index is built by using the EVRF combined with a blinded bloom filter, producing the resulting encapsulated search index.
- The built index is appended to the encrypted file.
- There is authorization needed to perform searches, similar to the authorization needed to decrypt the file.
- Let us assume the user wants to search "computer". The user enters the keyword they wish to search in the interface, as shown in the following figure:



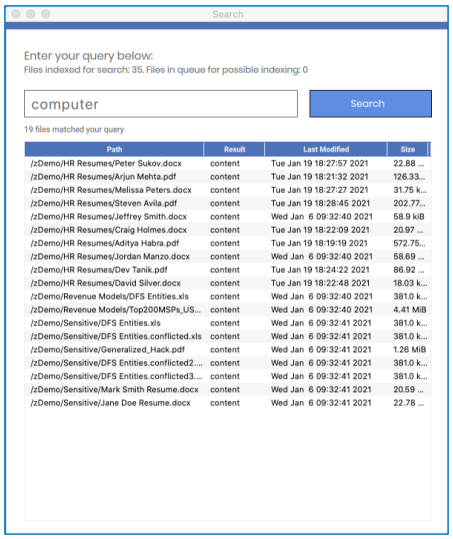

- This search needs to then be approved on a mobile device. The user's mobile device receives this prompt:



– When the user taps to approve, the mobile device computes a partial search token for each word of the search query. The partial search token is computed based on the share of the secret key and using its handle.

– This share of the search token is communicated to the software.

– The software then combines the shares it has received from each device to generate the actual search token. This token is then used to return the set of all files that contain the keyword. This is how a typical output looks:




Notice, different files have different sizes, but this does not affect the performance in practice.

– In the event a file needs to be transferred to another user, the handles can be re-derived using the algorithms of Delegated EVRF. The delegation step costs $O(f)$, where $f$ is the number of files.

# B Bilinear Groups and Hardness Assumptions

BILINEAR GROUPS. We use bilinear maps in our constructions.[15] We briefly review their properties below (see [25] for a more comprehensive treatment).

Consider two (multiplicative) cyclic groups $\mathbb{G}$ and $\mathbb{G}_1$ of prime order $p$. Let $g$ be a generator of $\mathbb{G}$. Roughly speaking, a mapping is **bilinear** if it is linear with respect to each of its variables:

**Definition 16.** *An (admissible) bilinear map is a map $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$ with the following properties:*

1. **Bilinear:** *for all $u, v \in \mathbb{G}$ and $x, y \in \mathbb{Z}$, we have $e(u^x, v^y) = e(u, v)^{xy}$.*
2. **Non-degenerate:** $e(g, g) \neq 1$.

[15] For simplicity of exposition, we will use symmetric bilinear maps. However, all our assumptions and constructions easily extend to the asymmetric variant; see [9].

3. **Computable:** *there is an efficient algorithm to compute* $e(u, v)$ *for all* $u, v \in \mathbb{G}$.

We say that a group $\mathbb{G}$ is bilinear if the group action in $\mathbb{G}$ is efficiently computable and there exists a group $\mathbb{G}_1$ and an admissible bilinear map $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$. Henceforth, we shall use $\mathbb{G}^*$ to stand for $\mathbb{G} \backslash \{1_{\mathbb{G}}\}$. Such maps can be constructed from Weil and Tate pairings on elliptic curves or abelian varieties [9, 27, 33].

DDH IS EASY IN BILENEAR GROUPS. Given two group elements $A = g^a, B = g^b \in \mathbb{G}$, we define the $\mathsf{DH}(A, B) = \mathsf{DH}(g^a, g^b) = g^{ab} \in \mathbb{G}$. Note that an admissible bilinear map provides an algorithm for solving the decisional Diffie-Hellman problem (DDH) in $\mathbb{G}$. Specifically, to determine whether $(g, A, B, R)$ is a *DDH tuple*, meaning check if $R = \mathsf{DH}(A, B)$. Indeed, $R = \mathsf{DH}(A, B)$ iff $e(A, B) = e(g, R)$, which is efficiently checkable in bilinear groups. We now state the hardness assumptions on which our constructions are based. In what follows, we let $\mathbb{G}$ be a bilinear group of prime order $p$, and let $g$ be its generator.

BILINEAR DIFFIE-HELLMAN ASSUMPTION. Our basic and unidirectional EVRF constructions will rely on the Bilinear Decisional Diffie-Hellman (BDDH) assumption from the original Boneh-Franklin paper [9]. The BDDH problem on $\mathbb{G}$ asks: given $\left(g, g^a, g^b, g^r\right) \in (\mathbb{G}^*)^4$ as input, distinguish the value $e(g, g)^{abr}$ from random element $h \leftarrow \mathbb{G}_1$. Formally, an algorithm $\mathcal{B}$ has advantage $\varepsilon$ in solving BDDH in $\mathbb{G}$ if

$$\left| \Pr\left[\mathcal{B}(g, g^a, g^b, g^r, e(g, g)^{abr}) = 1\right] - \Pr\left[\mathcal{B}(g, g^a, g^b, g^r, g_1) = 1\right] \right| \leq \varepsilon,$$

where the probability is over the internal coin tosses of $\mathcal{A}$ and the choice of $a, b, r \in \mathbb{Z}_p^*$ and $g_1 \in \mathbb{G}_1$.

**Definition 17.** *(*BDDH *assumption) We say that* BDDH assumption *holds in* $\mathbb{G}$ *if every PPT algorithm* $\mathcal{B}$ *has advantage at most* $\varepsilon(k) = negl(k)$ *in solving the* BDDH *problem in* $\mathbb{G}$.

EXTENDED BDDH ASSUMPTION. Our one-time bidirectional delegatable EVRF construction will rely on the *extended* Bilinear Decisional Diffie-Hellman (eBDDH) that we introduce, where the attacker is additionally given the value $g^{1/a} \in \mathbb{G}^*$. Formally, an algorithm $\mathcal{B}$ has advantage $\varepsilon$ in solving eBDDH in $\mathbb{G}$ if

$$\left| \Pr\left[\mathcal{B}(g, g^{1/a}, g^a, g^b, g^r, e(g, g)^{abr}) = 1\right] - \Pr\left[\mathcal{B}(g, g^{1/a}, g^a, g^b, g^r, g_1) = 1\right] \right| \leq \varepsilon,$$

where the probability is over the internal coin tosses of $\mathcal{A}$ and the choice of $a, b, r \in \mathbb{Z}_p^*$ and $g_1 \in \mathbb{G}_1$.

**Definition 18.** *(*eBDDH *assumption) We say that* extended eBDDH assumption *holds in* $\mathbb{G}$ *if every PPT algorithm* $\mathcal{B}$ *has advantage at most* $\varepsilon(k) = negl(k)$ *in solving the* eBDDH *problem in* $\mathbb{G}$.

INVERSION-ORACLE BDDH ASSUMPTION. Our multi-time bidirectional delegatable EVRF construction will rely on the *inversion-oracle* Bilinear Decisional Diffie-Hellman (iBDDH) that we introduce, where the attacker is additionally given the oracle $\mathcal{O}_a(h) = h^{1/a} \in \mathbb{G}^*$. Formally, an algorithm $\mathcal{B}$ has advantage $\varepsilon$ in solving iBDDH in $\mathbb{G}$ if

$$\left| \Pr\left[\mathcal{B}^{\mathcal{O}_a(\cdot)}(g, g^a, g^b, g^r, e(g, g)^{abr}) = 1\right] - \Pr\left[\mathcal{B}^{\mathcal{O}_a(\cdot)}(g, g^a, g^b, g^r, g_1) = 1\right] \right| \leq \varepsilon,$$

where the probability is over the internal coin tosses of $\mathcal{A}$ and the choice of $a, b, r \in \mathbb{Z}_p^*$ and $g_1 \in \mathbb{G}_1$.

**Definition 19.** *(*iBDDH *assumption) We say that* inversion-oracle BDDH assumption *(*iBDDH*) holds in* $\mathbb{G}$ *if every PPT algorithm* $\mathcal{B}$ *has advantage at most* $\varepsilon(k) = negl(k)$ *in solving the* iBDDH *problem in* $\mathbb{G}$.

Clearly, iBDDH assumption is stronger than eBDDH (as $g^{1/a} = \mathcal{O}_a(g)$), which in turn is stronger than the well believed BDDH assumption.

$$\mathsf{iBDDH} \implies \mathsf{eBDDH} \implies \mathsf{BDDH}$$

These assumptions can be proven secure in the Generic Group Model. We present an analysis of the iBDDH assumption in the generic group model in Section B.1.

## B.1 Generic Group Model Proof for iBDDH Assumption

In this section, we examine the iBDDH assumption in the generic group model [47].

In the generic group model, elements of $\mathbb{G}, \mathbb{G}_1$ are encoded as unique random strings. We define an injective function $\theta : \mathbb{Z}_p \mapsto \{0,1\}^*$ which maps $a \in \mathbb{Z}_p$ to the string encoding of the group element $g^a \in \mathbb{G}$. Similarly, we define $\theta_1 : \mathbb{Z}_p \mapsto \{0,1\}^*$ for $\mathbb{G}_1$. The encodings are such that non-group operations are meaningless. Typically, there exist three oracles - one which computes the group action in $\mathbb{G}$, another which computes the group action in $\mathbb{G}_1$, and a third that compute the bilinear pairing $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$. In our case, there exists a fourth oracle that models the inversion oracle $\mathcal{O}_a(\cdot)$.

**Theorem 20.** *Let $\mathcal{A}$ be an algorithm that solves the* iBDDH *problem. Assume that $a \in \mathbb{Z}_p^*$, $b, c, r \in \mathbb{Z}_p$, and the encoding functions $\theta, \theta_1$ are chosen at random. If $\mathcal{A}$ makes at most $q_G$ queries to the four oracles, then*

$$\left| \Pr\left[ \mathcal{A}^{\mathcal{O}_a(\cdot)} \begin{pmatrix} p, \theta(1), \\ \theta(a), \theta(b), \theta(c), \\ \theta_1(\Gamma_0), \theta_1(\Gamma_1) \end{pmatrix} = \beta \middle| \begin{array}{l} \beta \leftarrow \{0,1\}; \\ a \leftarrow \mathbb{Z}_p^*, b, c, r \leftarrow \mathbb{Z}_p, \\ \Gamma_\beta = abc, \Gamma_{1-\beta} = r \end{array} \right] - \frac{1}{2} \right| \leq \frac{2 \cdot (q_G + 6)^2 (q_G - 3)}{p}$$

*Proof.* Instead of letting $\mathcal{A}$ interact with the actual oracles, we play the following game.

We begin by maintaining two dynamic lists $L, L'$. Informally, $L$ contains the information that $\mathcal{A}$ has garnered about group elements in $\mathbb{G}$, while $L'$ contains information about those elements in $\mathbb{G}_1$. Formally, we have

$$L = \{(F_i, s_i) : i = 0, \ldots, t-1\},$$
$$L_1 = \{(F_i', s_i') : i = 0, \ldots, t'-1\}.$$

Here $s_i, s_i' \in \{0,1\}^*$ are random encodings and
$F_i, F_i' \in \mathbb{Z}_p[A, B, C, A^{-1}, \Gamma_0, \Gamma_1]$ are multivariate *Laurent* polynomials in $A, B, C, A^{-1}, \Gamma_0, \Gamma_1$. Note that this is a departure from typical modeling which models $F_i$ as polynomials with only positive degree elements. However, we need Laurent polynomials here to capture the inversion oracle.

In the beginning of the game, the lists are initialized as follows: $F_0 = 1, F_1 = A, F_2 = B, F_3 = C$, $F_0' = \Gamma_0, F_1' = \Gamma_1$. The corresponding encodings are set to arbitrary distinct strings in $\{0,1\}^*$. At this state, the lists have sizes $t = 4, t' = 2$. The total length of lists at a step $\tau \leq q_G$ in the game must satisfy:

$$t + t' = \tau + 6, \tag{4}$$

and it is easy to note that this condition is met at the start of the game.

We start the game by providing $\mathcal{A}$ with encodings $s_0, s_1, s_2, s_3, s_0', s_1'$. Algorithm $\mathcal{A}$ begins to make oracle queries and we respond as follows:

– Group Actions: $\mathcal{A}$ provides a multiply/divide bit and two operands $0 \leq i, j < t$ corresponding to two encodings $s_i, s_j$. We first compute $F_t = F_i \pm F_j$ depending on the choice of the bit. We then check to see if $\exists \ell < t$ such that $F_\ell = F_t$. If yes, we set $s_t = s_\ell$. Otherwise, we set $s_t$ to a random string in $\{0,1\}^* \setminus \{s_0, \ldots, s_{t-1}\}$. Then increment $t$ by 1 and return $s_t$ to $\mathcal{A}$. Similarly, we handle operations in $\mathbb{G}_1$, except that the operation is on the other list $L'$.

– Bilinear Pairing: $\mathcal{A}$ provides $0 \leq i, j < t'$ indicating operands $s_i, s_j$. First compute $F'_{t'} = F_i \cdot F_j$. We then check to see if $\exists \ell < t'$ such that $F'_\ell = F'_{t'}$. If yes, we set $s'_{t'} = s'_\ell$. Otherwise, we set $s'_{t'}$ to a random string in $\{0,1\}^* \setminus \{s'_0, \ldots, s'_{t'-1}\}$. Then increment $t'$ by 1 and return $s'_{t'}$ to $\mathcal{A}$.

– Inversion Oracle: $\mathcal{A}$ provides $0 \leq i < t$ indicating operand $s_i$. First compute $F_t = F_i/A$. We then check to see if $\exists \ell < t$ such that $F_\ell = F_t$. If yes, we set $s_t = s_\ell$. Otherwise, we set $s_t$ to a random string in $\{0,1\}^* \setminus \{s_0, \ldots, s_{t-1}\}$. Then increment $t$ by 1 and return $s_t$ to $\mathcal{A}$.

After making at most $q_G$ queries, $\mathcal{A}$ halts with a guess $\hat{\beta} \in \{0,1\}$. Then we choose $a \leftarrow \mathbb{Z}_p^*, b, c, r \leftarrow \mathbb{Z}_p$. Then, consider $\Gamma_\beta \leftarrow abc$, $\Gamma_{1-\beta} \leftarrow r$ for both choices of $\beta$. By sampling the values only after $\mathcal{A}$'s guess bit, the simulation does not reveal anything about $\beta$, unless our indeterminates give rise to some non-trivial equality relation for the sampled values. Specifically, $\mathcal{A}$ wins the game if for any $F_i \neq F_j$ or any $F'_i \neq F'_j$, one of the following conditions hold:

1. $F_i(a, b, c, a^{-1}, abc, r) - F_j(a, b, c, a^{-1}, abc, r) = 0$
2. $F_i(a, b, c, a^{-1}, r, abc) - F_j(a, b, c, a^{-1}, r, abc) = 0$
3. $F'_i(a, b, c, a^{-1}, abc, r) - F'_j(a, b, c, a^{-1}, abc, r) = 0$
4. $F'_i(a, b, c, a^{-1}, r, abc) - F'_j(a, b, c, a^{-1}, r, abc) = 0$

Let us look at the degree of the polynomials. For all $i$, $\deg(F_i) \leq t-3$, and $\deg(F'_i) \leq 2(t-3)$. In other words, each $F_i$ is a polynomial in $A, B, C, A^{-1}, ABC, R$ whose degree is at most $t-3$. Consider the polynomial: $f_i = F_i \cdot A^{t-4}$. Note that $f_i$ is a polynomial only in $A, B, C, ABC, R$ and $\deg(f') \leq t-3$. Further, every root of $F_i$ is a root of $f_i$ and there are at most $t-3$ roots of $f_i$. Similarly, consider the polynomial $f'_i = F'_i \cdot A^{2t-8}$. The degree of $f'_i$ is at most $2t-6$ and every root of $F'_i$ is a root of $f'_i$, and there are at most $2t-6$ roots of $f'_i$. We can therefore apply Schwartz-Zippel Lemma [45] to get that for all $i, j$, $\Pr[F_i - F_j = 0] \leq \Pr[f_i - f_j = 0] \leq (t-3)/p, \Pr[F'_i - F'_j] \leq \Pr[f'_i - f'_j = 0] \leq 2(t-3)/p$. Therefore, $\mathcal{A}$'s advantage is:

$$\begin{aligned}
\varepsilon &\leq 2 \cdot \left( \binom{t}{2} \frac{t-3}{p} + \binom{t'}{2} \frac{2 \cdot (t-3)}{p} \right) \\
&< (t + t')^2 \cdot \frac{2 \cdot (t-3)}{p} \\
&< (q_G + 6)^2 \cdot \frac{2 \cdot (t-3)}{p} \\
&< (q_G + 6)^2 \cdot \frac{2 \cdot (q_G - 3)}{p} = O\left( \frac{q_G^3}{p} \right)
\end{aligned}$$

□

# C Deferred Security Proofs

## C.1 Proof of Encapsulated Search Index security

**Theorem 6.** *If* EVRF *satisfies the* ***$-Core*** *property, then Encapsulated Search Index is* ***CCA*** *secure. Further, if the* EVRF *(resp.* DS *) is threshold and/or delegatable (resp. updatable; see Remark 7), the resulting ESI inherits the same.*

*Proof.* The proof is through a sequence of Hybrids where at each step we replace the index corresponding to a word $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$ with a random index, one by one. Recall that the process of building an index is to use the keyword as the input to the EVRF, making the output of the EVRF as the index. The key idea behind the proof is that only these words can help distinguish the two documents and the adversary is prevented from receiving computation of S-Prove on these words. Thus, the EVRF of these words is never realized by the adversary. Let $H_i$ be the distribution where the first $i$ distinguishing words in $D_1$ have been replaced by a random string and call the resulting encrypted index $E_i$. We will show that if $\mathcal{A}$ can distinguish between $H_i$ and $H_{i+1}$, then we can construct an adversary $\mathcal{B}$ that can win in the EVRF security game.

Formally, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker trying to distinguish between hybrids $H_i$ and $H_{i+1}$, having advantage $\varepsilon$. $\mathcal{A}$ is given public key $PK$ (for unknown $SK$) and oracle access to S-Prove$(SK, \cdot, \cdot)$. In addition, the challenge index is either $E_i$ or $E_{i+1}$.

Using this attacker $\mathcal{A}$, we now define a PPT attacker $\mathcal{B}$ which will break the $\mathbf{\$ - Core}$ property of $EVRF$. $\mathcal{B}$ is given the value $PK$, challenge ciphertext $C$, and oracle access to Prove. Note that S-Prove and Prove are identical functions for our construction.

Definition of $\mathcal{B}^{\text{Prove}(SK,\cdot,\cdot)}(PK, C)$.

- **Setup:** $\mathcal{B}$ provides to $\mathcal{A}$ the $PK$ value.
- **Queries to S-Prove**$(SK, \cdot, \cdot)$**:** $\mathcal{B}$ receives inputs $(c_i, w_i)$
  - Uses its access to Prove queries to receive the output of Prove$(SK, c_i, w_i) = z_i$.
  - Responds to $\mathcal{A}$ with $z_i$ after recording $(c_i, w_i, z_i)$ in a table $T$.
  - $\mathcal{B}$ uses table $T$ to verify that $\mathcal{A}$ is not querying S-Prove on a distinguishing word $w_i$.
- **Challenge Query:** $\mathcal{B}$ receives two documents $D_1, D_2$. It identifies the set of distinguishing words $W$, i.e., $W = (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$. Let $W = \{w'_1, \ldots, w'_m\}$. $\mathcal{B}$ tests the legality of $\mathcal{A}$ using $T$. If the test passes, then for $w \in D$, do the following:
  - If $w \in \{w_{i+2}, \ldots, w_m\}$ or $w \notin W$, then
    - Query oracle Prove with input $(C, w)$ to receive as output $z = \text{Core}(SK, \text{Split}(PK, C), w)$
    - Compute Post$(PK, z, C, x) = y$.
    - Add $y$ to $Y$.
  - If $w \in \{w_1, \ldots, w_i\}$, then:
    - Pick $y$ at random.
    - Add $y$ to $Y$,
  - If $w = w_{i+1}$, then:
    - $\mathcal{B}$ uses $w_{i+1}$ as its challenge word. It receives as response $y_{i+1}$.

* Add $y_{i+1}$ to $Y$.

Run DS.CONSTRUCT$(Y) = E^*$. Set $c^* = C$. Forward $(E^*, c^*)$ to $\mathcal{A}$.

– **Finish:** Let $\mathcal{A}$ return $b'$ as its guess. It forwards $b'$ as its guess.

ANALYSIS OF REDUCTION. If the challenger's bit was 0, then $\mathcal{B}$ simulates the distribution $H_i$ perfectly, where the first $i$ distinguishing words are indexed by a random value. If the challenger's bit was 1, then $\mathcal{B}$ simulates the distribution $H_{i+1}$ perfectly where the first $i+1$ distinguishing words are indexed by a random value. Therefore, the advantage of $\mathcal{B}$ in distinguishing between real or random value is same as $\mathcal{A}$'s advantage in distinguishing between hybrids $H_i$ and $H_{i+1}$ (which is $\varepsilon$). □

## C.2 Proof of EVRF security

**Theorem 7.** *The standard* EVRF *given in Construction 2 satisfies the* ***$*-Core*** *property under the* BDDH *assumption in the random oracle model.*

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker against the **$-Core** property of EVRF, having advantage $\varepsilon$. $\mathcal{A}$ is given the public key $A = g^a$ (for unknown $a$), challenge ciphertext $R = g^r$ (for unknown $r$), and oracle access to PROVE, which is equal to CORE due to empty SPLIT step. Namely, $\mathcal{A}$ has access to CORE$(a, \cdot, \cdot)$: on query $(R', x')$, $\mathcal{A}$ gets $z' = H(R', x')^a$. $\mathcal{A}$ then outputs challenge $x$, gets back a value $y$ which is either $e(H(R,x)^a, R) = e(H(R,x), g)^{ar}$ or uniform over $\mathbb{G}_1$, and has to tell which without asking its CORE$(a, \cdot, \cdot)$ oracle on the challenge $(R, x)$. Additionally, $\mathcal{A}$ expects to have oracle access to the random oracle $H : \{0,1\}^* \to \mathbb{G}$, and values $g$, $A = g^a, B = g^b, R = g^r$ (for unknown $a, b, r$), and a challenge $g_1$, which is either $e(g,g)^{abr}$ or uniform in $\mathbb{G}_1$. There is no random oracle for $\mathcal{B}$.

DEFINITION OF $\mathcal{B}(g, A, B, R, g_1)$. Run $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as follows:

– **Setup:** Pass $PK = A$ and challenge ciphertext $C = R$ to $\mathcal{A}_1$.

– **Queries to $H$:** let $q$ be the upper bound on the number of queries to CORE made by $\mathcal{A}$. $\mathcal{B}$ will maintain a table $T$ — initially empty — of the form $\{((R_i, x_i), \beta_i, coin_i)\}$, where $i$ is the index of the query to $H$ incremented with each query, $(R_i, x_i)$ in the input to the query (inputs not of this form are consistently answered at random), and the meaning of $\beta_i \in \mathbb{Z}_p$ and $coin_i \in \{0,1\}$ will be explained below:

1. If $(R_i, x_i)$ was already made, meaning there is $j < i$ such that $(R_i, x_i) = (R_j, x_j)$ and $((R_j, x_j), \beta_j, coin_j) \in T$, then respond with:

   $H(R_i, x_i) = g^{\beta_j}$ if $coin_j = 0$, and with $H(R_i, x_i) = B^{\beta_j}$ if $coin_j = 1$.
2. Otherwise, flip a fresh $coin_i \in \{0,1\}$ such that $\Pr[coin_i = 1] = 1/q$, sample random $\beta_i \leftarrow_r \mathbb{Z}_p$, add the tuple $((R_i, x_i), \beta_i, coin_i)$ to $T$, and respond with:

   $H(R_i, x_i) = g^{\beta_i}$ if $coin_i = 0$, and with $H(R_i, x_i) = B^{\beta_i}$ if $coin_i = 1$.

– **Queries to** CORE**:** Given query $(R', x')$ to CORE, check if $\mathcal{A}$ made the $H$-query $(R', x')$ by checking if there exists $j$ such that $(R', x') = (R_j, x_j)$ and $((R_j, x_j), \beta_j, coin_j) \in T$. If such query was not made, make this query for $\mathcal{A}$ following the normal $H$-query simulation strategy above. In either case, retrieve the record $((R_j, x_j), \beta_j, coin_j) \in T$ with $(R', x') = (R_j, x_j)$. Respond as follows:

  If $coin_j = 1$, abort the simulation, outputting a random bit $b' \in_r \{0, 1\}$.

  Otherwise, we know $H(R', x') = g^{\beta_j}$, so $\mathcal{B}$ outputs partial output $z' = A^{\beta_j}$.

  (Notice, $z' = A^{\beta_j} = g^{a\beta_j} = H(R', x')^a$, for unknown $a$.)

– **Challenge query:** When $\mathcal{A}_1$ produces challenge $x$, produce answer $y$ as follows. Check if $\mathcal{A}_1$ made the $H$-query $(R, x)$ by checking if there exists $j$ such that $(R, x) = (R_j, x_j)$ and $((R_j, x_j), \beta_j, coin_j) \in T$. If such query was not made, make this query for $\mathcal{A}$ following the normal $H$-query simulation strategy above. In either case, retrieve the record $((R_j, x_j), \beta_j, coin_j) \in T$ with $(R, x) = (R_j, x_j)$. Respond as follows:

  If $coin_j = 0$, abort the simulation, outputting a random bit $b' \in_r \{0, 1\}$.

  Otherwise, we know $H(R, x) = B^{\beta_j}$, and $\mathcal{B}$ will output challenge $y = g_1^{\beta_j}$.

  (Notice, $y_0 = \text{EVAL}(a, R, x) = e(H(R, x)^a, R) = e(B^{\beta_j a}, g^r) = (e(g, g)^{abr})^{\beta_j}$.)

– **Finish:** If the simulation did not fail, and $\mathcal{A}$ produces a guess $b'$, $\mathcal{B}$ outputs the same $b'$.

ANALYSIS OF THE REDUCTION. Assume the advantage of $\mathcal{A}$ is $\varepsilon$, and let us denote by $\varepsilon'$ the advantage of $\mathcal{B}$. We will argue that $\varepsilon' > \varepsilon/3q$, which would complete the proof, since $q$ is polynomial. Let use define "failure event" $F$ to be that $\mathcal{B}$ had to abort the simulation of $\mathcal{A}$, either during one of the CORE queries (if $coin_j = 1$), or when simulating the challenge (if $coin_j = 0$). Recall, if $F$ happens, $\mathcal{B}$ still outputs a random bit $b'$, which has $1/2$ chance to be equal to the challenge $b$. Let us also define "success probability" $\gamma \in [0, 1]$ as $\Pr[\neg F] = \gamma$. Notice,

$$\begin{aligned}
\frac{1}{2} + \varepsilon' &= \Pr[b = b'] \\
&= \Pr[F] \cdot \Pr[b = b' \mid F] + \Pr[\neg F] \cdot \Pr[b = b' \mid \neg F] \\
&= (1 - \gamma) \cdot \frac{1}{2} + \gamma \cdot \Pr[b = b' \mid \neg F] \\
&= \frac{1}{2} + \gamma \cdot \left( \Pr[b = b' \mid \neg F] - \frac{1}{2} \right)
\end{aligned}$$

Thus, to complete our proof that $\varepsilon' > \varepsilon/3q$, it suffices to show the following two claims:

**Claim 1.** $\gamma = \Pr[\neg F] > \frac{1}{3q}$.

**Claim 2** (2)**.** $\Pr[b = b' \mid \neg F] = \frac{1}{2} + \varepsilon$.

Proof of Claim 1. We see that in order for $\mathcal{B}$ not to abort the simulation, all random bits $coin_j$ have to be equal to 0 (each independently happens with probability $1-1/q$) when simulating at most $q$ Core queries, and also the "challenge" coin $coin_j$ should be 1 (independently happens with probability $1/q$, since none of the Core queries used the challenge $(R,x)$). This means that, overall,

$$\gamma \geq \left(1-\frac{1}{q}\right)^q \cdot \frac{1}{q} > \frac{1}{3q}$$

completing the proof. □

Proof of Claim 2. We claim that when $\mathcal{B}$ successfully completes the simulation, he perfectly simulates the run of $\mathcal{A}$. More precisely, in this case the run of $\mathcal{B}$ with challenge bit $b=0$ is identical to the run of $\mathcal{A}$ with challenge bit $b=0$, and the same for $b=1$. This will complete the proof, as then

$$\Pr[b=b' \mid \neg F] = \Pr[\mathcal{A} \text{ wins}] = \frac{1}{2}+\varepsilon$$

To see this, first we notice that all the queries to $H$ are answered at random irrespective of the value of the $coin_j$, since both distributions $g^{\beta_j}$ and $B^{\beta_j}$ are perfectly uniform when $\beta_j \leftarrow_r \mathbb{Z}_p$.

Second, we already saw that when all Core queries are answered, each partial answer $z'$ is correct, as $z' = A^{\beta_j} = g^{a\beta_j} = H(R',x')^a$ (for correct, although unknown, $a$). Finally, let use look at the challenge query $(R,x)$. Since the simulation succeeded, we know $H(R,x) = B^{\beta_j}$, for some fresh and random $\beta_j \in \mathbb{Z}_p$. This means the correct output $y_0 = \textsc{Eval}(a,R,x) = e(H(R,x)^a, R) = e(B^{\beta_j a}, g^r) = (e(g,g)^{abr})^{\beta_j}$. Our simulator $\mathcal{B}$ responded with $g_1^{\beta_j}$, where $g_1$ is its own challenge. Thus:

When $b=0$, we have $g_1 = e(g,g)^{abr}$, meaning that $y_0 = g_1^{\beta_j}$ indeed.

When $b=1$, $g_1 \in_r \mathbb{G}_1$, which means that the response $g_1^{\beta_j}$ is identically distributed with a uniform answer $y_1 \in_r \mathbb{G}_1$, as needed.

This completes the proof of Claim 1 and Theorem 7. □

### C.3 Proof of TEVRF security

**Theorem 9.** *If Construction 2 satisfies the* ***$-Core*** *property of standard* EVRF*, then Construction 3 satisfies the* ***$-DCore*** *property of threshold* EVRF*. By Theorem 7, it follows that Construction 3 satisfies the* ***$-DCore*** *property under the* BDDH *assumption in the random oracle model.*

*Proof.* Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker against the **$-DCore** property of TEVRF, having advantage $\varepsilon$. $\mathcal{A}$ first chooses $t-1$ indices $S = \{i_1, \ldots, i_{t-1}\}$ where each index is a subset of $\{1,\ldots,n\}$. Then, $\mathcal{A}$ is given the public key $A = g^a$ (for unknown $a$), challenge ciphertext $R = g^r$ (for unknown $r$), secret keys $sk_{i_1} = a_{i_1} = f(i_1), \ldots, sk_{i_{t-1}} = a_{i_{t-1}} = f(i_{t-1})$ (for unknown polynomial $f$ of degree $t$ such that $f(0) = a$), verification keys $\mathbf{VK}$, and oracle access to Prove, with equal valued Split step. Namely, $\mathcal{A}$ has access to $\textsc{Prove}(sk_i,\cdot,\cdot)$: on query $(i,R',x')$, $\mathcal{A}$ gets $z' = H(R',x')^{a_i}$. $\mathcal{A}$ then outputs challenge $x$, gets back a value $y$ which is either $e(H(R,x)^a, R) = e(H(R,x),g)^{ar}$ or uniform over $\mathbb{G}_1$, and has to tell which without asking its $\textsc{Prove}(\mathbf{SK},\cdot,\cdot,\cdot)$ oracle on input $(j,R,x)$ for $j \notin S$. Additionally, $\mathcal{A}$ expects to have oracle access to the random oracle $H : \{0,1\}^* \to \mathbb{G}$.

Using this attacker $\mathcal{A}$, we now define a PPT attacker $\mathcal{B}$ which will break the **$-Core** property of EVRF. $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is given the values $g$, public key $A^* = g^{a^*}$ (for unknown $a^*$), ciphertext $R^* = g^{r^*}$ (for unknown $r^*$), and an oracle access to Prove which is equal to Core due to empty Split step. Namely $\mathcal{B}$ has access to $\textsc{Core}(a^*,\cdot,\cdot)$ on query $(R',x')$, it receives

in response $H(R', x')^{a^*}$. $\mathcal{B}$ then outputs a challenge $x$ and receives a response $y$ which is either $e(H(R^*, x)^{a^*}, R^*) = e(H(R^*, x), g)^{a^* r}$ or uniform in $\mathbb{G}_1$. Additionally, $\mathcal{B}$ expects to have oracle access to the random oracle $H : \{0,1\}^* \to \mathbb{G}$.

Definition of $\mathcal{B}^{\text{Core}(a^*,\cdot,\cdot),H(\cdot,\cdot)}(A^*, R^*)$. Run $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ as follows:

- **Setup:** $\mathcal{B}$ does the following during Setup.
  - Receive set $S = \{i_1, \ldots, i_{t-1}\}$ from $\mathcal{A}$.
  - Next $\mathcal{B}$ generates the key shares and public key as follows:
    - Sample $a_{i_1}, \ldots, a_{i_{t-1}} \in \mathbb{Z}_p$.
    - Pick $i_t \notin S$ at random.
    - Let $f \in \mathbb{Z}_p[X]$ be the degree $t-1$ polynomial implicitly defined to satisfy $f(i_t) = a^*$, and $f(i_j) = a_{i_j}$ for $j = 1, \ldots, t-1$.
    - Note that $\mathcal{B}$ does not know $f$ since it does not know $a^*$.
    - Recall that $f(0) = a$ and $A = g^a$ is the public key. $\mathcal{B}$ determines the Lagrange coefficients $\lambda_1, \ldots, \lambda_{t-1}, \lambda_t \in \mathbb{Z}_p$ such that $f(0) = \sum_{j=1}^{t} \lambda_j \cdot f(i_j)$. Note that these do not require the knowledge of $f$. Therefore, we can now compute $A = \prod_{j=1}^{t-1} g^{a_{i_j} \lambda_j} \cdot (A^*)^{\lambda_t}$.
    - $\mathcal{B}$ gives $a_{i_1}, \ldots, a_{i_{t-1}}, A$ to $\mathcal{A}$.
  - Next $\mathcal{B}$ computes the verification key $\mathbf{VK} = (vk_1, \ldots, vk_n)$ as follows:
    - For $i_j \in S$, this is easy and merely sets $vk_{i_j} = g^{a_{i_j}}$. Further it sets $vk_{i_t} = A^*$.
    - For $\ell \notin S$, $\mathcal{B}$ determines the Lagrange coefficients $\lambda'_1, \ldots, \lambda'_t$ such that $f(j) = \sum_{j=1}^{t} \lambda_j \cdot f(i_j)$. Again, this does not need knowledge of $f$. Now, it can set $vk_\ell = \prod_{j=1}^{t} vk_{i_j}^{\lambda'_j}$.
    - $\mathcal{B}$ gives $\mathbf{VK}$ to $\mathcal{A}$.
  - $\mathcal{B}$ also provides $R^*$ to $\mathcal{A}$.
- **Queries to H:** $\mathcal{B}$ merely responds to all queries from $\mathcal{A}$ to $H$ by using its oracle access to $H$.
- **Queries to Prove:** $\mathcal{B}$ will maintain a table $T$, which is initially empty, of the form $\{(j_k, R_k, x_k), w_k\}$ where $k$ is the index of the query to Prove incremented with each query $(j_k, R_k, x_k)$. On receiving a query $(j_k, R_k, x_k)$, it does the following:
  - If there exists $\ell < k$ such that $j_k = j_\ell, R_k = R_\ell, x_k = x_\ell$, then set $w_k = w_\ell$.
  - Else if $j_k \in S$, then it merely uses its oracle access to $H$ to receive the value $H(R_k, x_k) = h_k$. It then sets $w_k = h_k^{a_{j_k}}$.
  - Else if $j_k = i_t$, then it uses it oracle access to Core to receive $w_k$ which is actually equal to $H(R_k, x_k)^{a^*}$.
  - For all other $j_k$, it does the following:
    - Uses its oracle access to $H$ to receive the value $h_k = H(R_k, x_k)$.
    - Further uses its oracle access to Core to receive the value $h^*$ which is actually $H(R_k, x_k)^{a^*}$.

* Determines Lagrange coefficients $\lambda_1, \ldots, \lambda_t$ such that $f(j_k) = \sum_{\ell=1}^{t} \lambda_\ell \cdot f(i_\ell)$. Now, it computes $w_k = (h^*)^{\lambda_t} \cdot \prod_{\ell=1}^{t-1} h_k^{a_{i_\ell} \cdot \lambda_\ell}$

– Record $((j_k, R_k, x_k), w_k)$ and return $w_k$ to $\mathcal{A}$.

– **Challenge Query:** On receiving the challenge input $x$, $\mathcal{B}$ does the following:

– Check if there exists $((j_k, R_k, x_k), w_k) \in T$ such that $j_k \notin S$, $R_k = R^*$, and $x_k = x$. If yes, abort as $\mathcal{A}$ has violated the definition of the game.

– If not, $\mathcal{B}$ issues its challenge input as $x$. It receives $y^*$ which is either $e(H(R^*, x), R^*)^{a^*}$ or a random element in $\mathbb{G}_1$.

– It also uses its oracle access to $H$ to receive $h = H(R^*, x)$.

– Determines Lagrange coefficients $\lambda_1, \ldots, \lambda_t$ such that $f(0) = \sum_{\ell=1}^{t} \lambda_\ell \cdot f(i_\ell)$.

– Computes $z^* = \prod_{\ell=1}^{t-1} h^{\lambda_\ell \cdot a_{i_\ell}}$.

– It finally computes $y = y^{*\lambda_t} \cdot e(z^*, R^*)$ and outputs $y$ to $\mathcal{A}$

– **Finish:** It forwards $\mathcal{A}$'s guess as its own guess.

Analysis of the Reduction. When $y^* = e(H(R^*, x), R^*)^{a^*}$, then we get that:

$$\begin{aligned}
y &= y^{*\lambda_t} \cdot e(z^*, R^*) \\
&= e(H(R^*, x), R^*)^{a^* \cdot \lambda_t} \cdot e(z^*, R^*) \\
&= e(h, R^*)^{a^* \cdot \lambda_t} \cdot e(\prod_{\ell=1}^{t-1} h^{\lambda_\ell \cdot a_{i_\ell}}, R^*) \\
&= e(h, R^*)^{f(i_t) \cdot \lambda_t} \cdot e(h^{\sum_{\ell=1}^{t-1} \lambda_\ell \cdot f(i_\ell)}, R^*) \\
&= e(h, R^*)^{\sum_{\ell=1}^{t} \lambda_\ell \cdot f(i_\ell)} = e(h, R^*)^{f(0)} = e(h, R^*)^{a}
\end{aligned}$$

It is easy to see that if $y^*$ was a random group element in $\mathbb{G}_1$, the final output $y$ is also a random group element. Therefore, $\mathcal{B}$ perfectly simulates the $\mathbf{\$ - Core}$ game for $\mathcal{A}$, and $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s advantage. This completes the proof of Theorem 9. □

### C.4 Proof of Basic Delegation Security

**Theorem 11.** *The basic delegatable EVRF, given in Construction 4, satisfies the* ***Basic-$-Core*** *property under the* BDDH *assumption in the random oracle model.*

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker against the **Basic-$-Core** property of $\mathrm{DEVRF}_1$, having advantage $\varepsilon$. $\mathcal{A}$ is given the public key $A_1 = g^{a_1}$ (for unknown $a_1$), challenge ciphertext $C_1 = (A_1, R_1, R_1)$ (where $R_1 = g^r$ for unknown $r$), and has oracle access to 4 oracles: random oracle $H$, registration oracle Reg, honest evaluation oracle HProve, and honest delegation oracle HDel.

Legality of $\mathcal{A}$. Let $u$ be the polynomial upper bound on the number of overall honest public/secret keys (including the challenge) used by $A$ (meaning $\mathcal{A}$ made at most $u - 1$ registration queries). Denote corresponding key pairs $\{(a_i, A_i = g^{a_i})\}_{i=1}^{u}$, so that $a_1 = a$. Define $D_i = R_1^{a_1/a_i}$ and $C_i = (A_1, R_1, D_i)$, for $i = 1 \ldots u$. The legality condition on $\mathcal{A}$ states what $\mathcal{A}$ is not able to

call HPROVE$(i, C'_i, x)$, where $x$ is the challenge input produced by $\mathcal{A}_1$, for any $C'_i = (A'_i, R'_i, D'_i)$ where SAME$(A_1, C_1, A_i, C'_i) = 1$. This means $(A'_i, R'_i) = (A_1, R_1)$ and $e(A_1, R_1) = e(A_i, D'_i)$. It follows that $D'_i = D_i = R_1^{a_1/a_i}$ and overall $C'_i = C_i$. Hence, in our reduction to BDDH we can assume that $\mathcal{A}$ never calls HPROVE$(i, (A_1, R_1, D_i), x)$, for any $i \in [u]$. Moreover, for any $D'_i \neq D_i$, a call HPROVE$(i, (A_1, R_1, D'_i), x)$ returns $\perp$, since the value $D'_i$ does not pass the delegation check: $e(A_1, R_1) = e(A_i, D_i) \neq e(A_i, D'_i)$. To sum up this discussion, without loss of generality in our reduction to BDDH, we can assume that

**Condition (*)**: *$\mathcal{A}$ never calls* HPROVE$(i, (A_1, R_1, D), x)$*, for any $i \in [u]$ and any $D$*

Namely, these calls either immediately return $\perp$ (if $D \neq D_i$, and this can be checked by $\mathcal{A}$), or disallowed anyway.

OUR REDUCTION. Our reduction, which uses $\mathcal{A}$ to build a BDDH attacker $\mathcal{B}(g, A, B, R, g_1)$, will proceed very similarly to the proof of Theorem 7, but with several small modifications:

1. **Initialization.** $\mathcal{B}$ will set the challenge key $A_1 = A$ (implicitly keeping the secret key $a_1 = a$ unknown), and the challenge ciphertext is $C_1 = (A, R, R)$, so we set $R_1 = D_1 = R$.

2. **Registration Oracle Reg.** $\mathcal{A}$ can set some polynomial number of honest key pairs $\{(A_i, a_i)\}_{i=2}^{u}$, where $A_i = g^{a_i}$. In our reduction, we set each such $A_i = A^{\alpha_i}$, for random $\alpha_i \in \mathbb{Z}_p$ chosen by $\mathcal{B}$. Since the $\alpha_i$'s are random, these keys are correctly distributed. For ease of notation we set $\alpha_1 = 1$.
   (Note, even though $\mathcal{B}$ does not know any of the secret keys $a_i = \alpha_i a \bmod p$, $\mathcal{B}$ can compute the ratio $a_i/a_j = \alpha_i/\alpha_j$.)

3. **Honest Delegation Oracle HDel.** When $\mathcal{A}$ calls
   HDEL$(i, (A', R', D'), j)$, $\mathcal{B}$ can perform the delegation check $e(A', R') \stackrel{?}{=} e(A_i, D')$ itself, and then can compute $(D')^{a_i/a_j} = (D')^{\alpha_i/\alpha_j}$ without the knowledge of any of the $a_i$'s.

4. **Hash Queries to $H$:** previously, the oracle $H$ used by the CORE procedure was only evaluated on the values $(R', x')$ given as input, but now we evaluate $H$ on the tuple $(A', R', x')$. Indeed, we already observed a simple attack showing that the construction is insecure if we only hash the value $(R', x')$. Nevertheless, our simulation of $H$ remains unchanged, modulo now accepting $(A', R', x')$ as input to $H$, rather than only $(R', x')$. Namely, every such fresh evaluation $H(A', R', x')$ chooses values $(\beta_i, coin_i)$, where $i$ is the query index, as in the proof of Theorem 7, and still sets

   $$H(A', R', x') = g^{\beta_i} \text{ if } coin_i = 0, \text{ and } H(A', R', x') = B^{\beta_i} \text{ if } coin_i = 1.$$

5. **Honest Prove Oracle HProve.** The oracle
   HPROVE$(i, (A', R', D'), x')$ first checks that $e(A', R') = e(A_i, D')$, and then proceeds as before in the proof of Theorem 7. In particular, it aborts if the value of $coin_j$ corresponding to the oracle query $H(A', R', x')$ is 1, and otherwise (meaning $H(A', R', x') = g^{\beta'}$ for some random $\beta'$) returns $z' = A_i^{\beta'}$, as before. This is correct since $z' = A_i^{\beta'} = g^{a_i \beta'} = H(A', R', x')^{a_i}$ (for unknown $a_i$).

6. **Challenge value $y$.** This is returned as before, by evaluating $H(A_1, R_1, x)$ and aborting if the value of $coin_j$ corresponding to the oracle query $H(A_1, R_1, x)$ is 0 (meaning $H(A_1, R_1, x) = B^{\beta}$ for some random $\beta$ if we do not abort). The challenge value $y$ is then set to $g_1^{\beta}$, as before.

7. **Finishing.** If the simulation succeeds until the end, $\mathcal{B}$ outputs the same $b'$ as $\mathcal{A}$.

Analysis of Reduction. We claim that the proof of the security of this reduction can go exactly as in Theorem 7. The only subtle point comes in the proof of Claim 1, where we argue that the values $(\beta', coin')$ sampled during the emulation of the evaluation oracle $\text{HProve}(i, (A', R', D'))$ are chosen *independently* from the value $(\beta, coin)$ used to emulate the challenge query $(A_1, R_1, x)$. This is indeed essential since we want all former coins to the 0, and the latter coin to be 1.

Fortunately, it immediately follows from Condition (*), as all evaluation queries on challenge $x$, must use $(A', R') \neq (A_1, R_1)$. Thus, we never have a conflict, and the proof of Claim 1 holds.

## C.5 Proof of Unidirectional Delegation Security

**Theorem 12.** *The delegatable EVRF given in Construction 5 satisfies the* ***Uni-$-Core*** *property under the* BDDH *assumption in the random oracle model.*

let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker against the **Uni-$-Core** property of $\text{DEVRF}_2$, having advantage $\varepsilon$. $\mathcal{A}$ is given the public key $A_1 = g^{a_1}$ (for unknown $a_1$), challenge ciphertext $C_1 = (A_1, R_1, R_1, \sigma_1)$ (where $R_1 = g^r$ for unknown $r$ and $\sigma_1 = H'(A_1, R_1)^{r_1}$), and has oracle access to 6 oracles: random oracles $H$, $H'$, registration oracle Reg, honest evaluation oracle HProve, honest delegation oracle HDel, and "OUT" delegation oracle OutDel. Note, oracles $H'$ and OutDel are new compared to the the proof of Theorem 11. Still, our proof will mimic almost exact the proof of Theorem 11, so we will use the same notation as in that proof, and only mention the key differences in our reduction.

Legality of $\mathcal{A}$: As before, we denote $u-1$ honest keys by $\{(a_i, A_i = g^{a_i})\}_{i=2}^{u}$, and define $D_i = R_1^{a_1/a_i}$, $C_i = (A_1, R_1, D_i, \sigma_1)$, for $i = 1 \ldots u$. The legality condition on $\mathcal{A}$ states what $\mathcal{A}$ is not able to call $\text{HProve}(i, C_i', x)$ or $\text{OutDel}(i, C_i', *)$, where $x$ is the challenge input produced by $\mathcal{A}_1$, for any $C_i' = (A_i', R_i', D_i', \sigma')$ where $\text{Same}(A_1, C_1, A_i, C_i') = 1$. This means $(A_i', R_i', \sigma') = (A_1, R_1, \sigma_1)$ and $e(A_1, R_1) = e(A_i, D_i')$. But this means $D_i' = D_i = R_1^{a_1/a_i}$ and overall $C_i' = C_i$. Moreover, as in the the proof of Theorem 11, we can assume without loss of generality that $\mathcal{A}$ will not use ciphertext $C_i' = (A_1, R_1, D', \sigma')$ for any $(D', \sigma') \neq (D_i, \sigma_1)$, as those will not pass the delegation or the signature check of either HProve or OutDel. Hence, similar to the proof of Theorem 11, we can we can assume that

**Condition (**)**: *$\mathcal{A}$ never calls* $\text{HProve}(i, (A_1, R_1, D, \sigma), x)$ *or* $\text{OutDel}(i, (A_1, R_1, D, \sigma), *)$, *for any $i \in [u]$ and any $D, \sigma$.*

Our Reduction. Our reduction, which uses $\mathcal{A}$ to build a BDDH attacker $\mathcal{B}(g, A, B, R, g_1)$, will proceed very similarly to the proof of Theorem 11, but with several small modifications:

1. **Hash Queries to $H'$:** $\mathcal{B}$ will maintain a table $T'$ containing entries of the form $(A', R', \gamma)$, where $A', B' \in \mathbb{G}$ and $\gamma \in \mathbb{Z}_p$. To create the first such entry in $T'$, $\mathcal{B}$ chooses a random value $\gamma_1 \in_r \mathbb{Z}_p$, and stores the tuple $(A, R, \gamma_1)$, where $A$ and $R$ come from $\mathcal{B}$'s challenge. After $T'$ is initialized, as above, all future queries $H'(A', R')$ are answered as follows.

   If $(A', R') = (A, R)$, respond with $g^{\gamma_1}$, meaning that we set $H'(A, R) = g^{\gamma_1}$.

   Otherwise, check if $T'$ has an entry of the form $(A', R', \gamma')$. If not, pick a random $\gamma' \leftarrow_r \mathbb{Z}_p$ and add the tuple $(A', R', \gamma')$ to $T'$. In either case, return $H'(A', R') = (A')^{\gamma'}$.

2. **Initialization.** $\mathcal{B}$ will set the challenge key $A_1 = A$ (implicitly keeping the secret key $a_1 = a$ unknown), and the challenge ciphertext is $C_1 = (A_1, R_1, D_1, \sigma_1)$, where $R_1 = D_1 = R$, and signature $\sigma_1 = R^{\gamma_1}$.
(Note, since $H'(A_1, R_1) = g^{\gamma_1}$, we have that $\sigma_1 = R^{\gamma_1} = (g^r)^{\gamma_1} = (g^{\gamma_1})^r = H'(A_1, R_1)^r$.)

3. **Registration Oracle Reg.** Same as the proof of Theorem 11. In particular, each public key $A_i = A^{\alpha_i}$, for random $\alpha_i \in \mathbb{Z}_p$ chosen by $\mathcal{B}$, and we set $\alpha_1 = 1$. This means that even though $\mathcal{B}$ does not know any of the secret keys $a_i = \alpha_i a \bmod p$, $\mathcal{B}$ can compute the ratio $a_i/a_j = \alpha_i/\alpha_j$.

4. **Honest Delegation Oracle HDel.** When $\mathcal{A}$ calls
$\textsc{HDel}(i, (A', R', D', \sigma'), j)$, $\mathcal{B}$ can perform the delegation check $e(A', R') \stackrel{?}{=} e(A_i, D')$ and the signature check
$e(H'(A', R'), R') \stackrel{?}{=} e(\sigma', g)$ himself (by evaluating $H'$ according to the standard strategy above). Then, $\mathcal{B}$ then can compute $(D')^{a_i/a_j} = (D')^{\alpha_i/\alpha_j}$ without the knowledge of any of the $a_i$'s.

5. **"OUT" Delegation Oracle OutDel.** When $\mathcal{A}$ calls
$\textsc{OutDel}(i, (A', R', D', \sigma'), a^*)$, for any secret key $a^*$, $\mathcal{B}$ can perform the delegation check $e(A', R') \stackrel{?}{=} e(A_i, D')$ and the signature check $e(H'(A', R'), R') \stackrel{?}{=} e(\sigma', g)$ himself (by evaluating $H'$ according to the standard strategy above). Moreover, from Condition (**) we know that $(A', R') \neq (A_1, R_1)$. Thus, when we set the value $H'(A', R')$ in our simulation of $H'$, we set it to $H'(A', R') = (A')^{\gamma'}$ for some random $\gamma'$ known to $\mathcal{B}$. $\mathcal{B}$ will combine this $\gamma'$ with the values of signature $\sigma'$ and secret key $a^*$ given by $\mathcal{A}$, and return

$$\textsc{OutDel}(i, (A', R', D', \sigma'), a^*) := (\sigma')^{(a^*\gamma')^{-1} \bmod p}$$

To check that this value indeed equals to $(D')^{a_i/a^*}$, it suffices to prove that $(D')^{a_i} = (\sigma')^{1/\gamma'}$. To see that, our signature check implies that

$$e(H'(A', R'), R') = e(\sigma', g) \implies e((A')^{\gamma'}, R') = e(\sigma', g)$$

This further implies that:

$$e(A', R') = e((\sigma')^{1/\gamma'}, g)$$

Finally, combined with the delegation check $e(A', R') = e(A_i, D')$ and $A_i = g^{a_i}$, we get that

$$e((\sigma')^{1/\gamma'}, g) = e(A', R') = e(A_i, D') = e((D')^{a_i}, g)$$

which implies that

$$(D')^{a_i} = (\sigma')^{1/\gamma'}$$

6. **Hash Queries to $H$:** These are identical to the proof of Theorem 11.

7. **Honest Evaluation Oracle HProve.** The oracle
$\textsc{HProve}(i, (A', R', D', \sigma'), x')$ first checks that $e(A', R') = e(A_i, D')$ and $e(H'(A', R'), R') = e(\sigma', g)$ (by evaluating $H'$ according to the standard strategy above). After that, it proceeds exactly like the proof of Theorem 11.

8. **Challenge value $y$.** This is returned as before, by evaluating $H(A_1, R_1, x)$ and aborting if the value of $coin_j$ corresponding to the oracle query $H(A_1, R_1, x)$ is 0 (meaning $H(A_1, R_1, x) = B^\beta$ for some random $\beta$ if we do not abort). The challenge value $y$ is then set to $g_1^\beta$, as before.

9. **Finishing.** If the simulation succeeds until the end, $\mathcal{B}$ outputs the same $b'$ as $\mathcal{A}$.

Analysis of Reduction. The analysis of the reduction then goes exactly as in Theorem 11 (which in turn is based on that in Theorem 7). As before, the only subtle point comes in the proof of Claim 1, where we argue that the emulation of $H$ during the evaluation queries Prove does not conflict with its emulation for the challenge query $H(A_1, R_1, x)$.

Fortunately, this immediately follows from our Condition (**) above, as all evaluation queries on challenge $x$ must use $(A', R') \neq (A_1, R_1)$. Thus, we never have a conflict, and the proof of Claim 1 holds.

### C.6 Proof of Bidirectional Delegation Security

**Theorem 13.** *The delegatable EVRF given in Construction 5 satisfies the* ***Bi-$-Core*** *property under the interactive* iBDDH *assumption in the random oracle model. It satisfies the strongest possible legality condition for the attacker (see Definition 21).*

We start by showing **Bi-$-Core** delegation security of $\text{DEVRF}_2$ as defined in Definition 10. Since the construction we analyze is identical to that of Theorem 12, our proof will be almost identical as well. Now, however, we also need to show how to simulate the "IN" delegation oracle $\text{InDel}(a^*, (A', R', D', \sigma'), i)$. This appears hard, since the values $a^*, A', R', D', \sigma'$ are adversarial, while the answer expected by the attacker should be equal (assuming the ciphertext is well formed) to $(D')^{a^*/a_i}$. Recall also that in our simulation we set $a_i = \alpha_i a$, where $\alpha_i$ were known by $\mathcal{B}$, but $a$ was unknown. Thus, our reduction $\mathcal{B}$ must be able to compute the value

$$D = (D')^{a^*/\alpha_i a} = ((D')^{a^*/\alpha_i})^{1/a}$$

Fortunately, we are now reducing from the inversion-oracle BDDH (iBDDH) assumption, given in Section B. Namely, in addition to its standard BDDH inputs, $\mathcal{B}$ also has oracle access to $\mathcal{O}_a(h) = h^{1/a}$. With this access, the simulation of
$\text{InDel}(a^*, (A', R', D', \sigma'), i)$ becomes trivial: $\mathcal{B}$ simply returns
$\mathcal{O}_a((D')^{a^*/\alpha_i}) = (D')^{a^*/a_i}$. This completes the first part of our proof. □

Stronger Legality Condition. We now show that under the iBDDH assumption we can actually strengthen the legality condition of $\mathcal{A}$ to be optimal; informally, only trivially delegated/evaluated ciphertexts can be broken by $\mathcal{A}$. We formalize this below.

**Definition 21.** *If $\mathcal{A}$ makes a call $C' = \text{HDel}(i, C, j)$, we say $\mathcal{A}$ creates a* delegation edge *from $(i, C)$ to $(j, C')$, denoted $(i, C) \to (j, C')$. A sequence of delegation edges $(i_1, C_1) \to (i_2, C_2) \to \ldots \to (i_t, C_t)$ defines a* delegation path *from $(i_1, C_1)$ to $(i_t, C_t)$, denoted $(i_1, C_1) \rightsquigarrow (i_t, C_t)$.*
*We now define a stronger legality condition on $\mathcal{A}$ (focusing on the bidirectional splittable case):*

- *Given challenge ciphertext $C_1$ on user 1, $\mathcal{A}$ produced no delegation path $(1, C_1) \rightsquigarrow (i, C')$, followed by either a call to $\text{HProve}(i, C', x)$, or a call to $\text{OutDel}(i, C', *)$, where $x$ is the challenge input returned by $\mathcal{A}_1$.*

Thus, the only "prohibited" ciphertexts $C'$ must have been explicitly obtained by the attacker. In contrast, our original Definition 10 only prevented $C'$ directly satisfying $\textsc{Same}(PK_1, C_1, PK_i, C') = 1$. While reasonable in many situations, the "gap" between the two notions involves an attacker who managed to find a ciphertext $C'$ satisfying $\textsc{Same}(PK_1, C_1, PK_i, C') = 1$ *without* creating a delegation path $(1, C_1) \rightsquigarrow (i, C')$.

Indeed, we saw that our unidirectional construction $\text{DEVRF}_2$ was trivially insecure wrt the stronger (and, clearly, optimal) legality condition on $\mathcal{A}$. However, when looking at our current construction $\text{DEVRF}_2$, we see that the existence of the signature $\sigma$ in the ciphertext appears to foil the trivial attack from Section 6.2. In fact, we show that is not luck, but the construction is actually secure w.r.t. to this stronger legality condition. Unfortunately, for this, we must rely on the stronger iBDDH assumption (even for basic or unidirectional security, so we might as well get the optimal bidirectional security).

Stronger Legality for $\text{DEVRF}_2$. Consider any attacker $\mathcal{A}$ against **Bi-$-Core** security of $\text{DEVRF}_2$ which satisfies the stronger legality condition, and has advantage $\varepsilon$. Recalling all the notation we used in the proof of Theorem 12, the challenge ciphertext $C_1 = (A_1, R_1, D_1, \sigma)$, where $D_1 = R_1$, and $u$ "special" ciphertexts which we cannot fully handle (see below) in our reduction are $C_i = (A_1, R_1, D_i, \sigma)$, where $D_i = R_1^{a_i/a_1}$. Indeed, these are the only ciphertexts satisfying

$$\textsc{Same}(A_1, C_1, A_i, C_i) = 1$$

Define the event $G$ to denote the "gap" between the two legality conditions of $\mathcal{A}$; namely,

- Let $G$ be the event that $\mathcal{A}$ made a call to $\textsc{HProve}(i, C_i, x)$ or $\textsc{OutDel}(i, C_i, *)$ for some $i \in [u]$, where $x$ is the challenge input produced by $\mathcal{A}_1$, but did not create a delegation path $(1, C_1) \rightsquigarrow (i, C_i)$.

Expanding the definition of $\mathcal{A}$'s advantage $\varepsilon$, we get that

$$\begin{aligned} \frac{1}{2} + \varepsilon &= \Pr[(b' = b) \wedge G] + \Pr[(b' = b) \wedge \neg G] \\ &\leq \Pr[G] + \Pr[(b' = b) \wedge \neg G] \end{aligned}$$

Thus, to prove that $\varepsilon = \text{negl}(k)$ under the iBDDH assumption, it suffices to prove the following two Lemmas:

**Lemma 22.** *Under the* iBDDH *assumption,* $\Pr[G] = negl(k)$.

**Lemma 23.** *Under the* iBDDH *assumption,* $\Pr[(b' = b) \wedge \neg G] \leq \frac{1}{2} + negl(k)$.

Lemma 23 is exactly the proof of security of $\text{DEVRF}_2$ we just finished at the beginning of this section above, as this corresponds to the run of $\mathcal{A}$ satisfying the original legality condition. Thus, to show the security of $\text{DEVRF}_2$ under the stronger legality condition, we only need to prove Lemma 22.

Proof of Lemma 22. Let use call a query $Q$ of $\mathcal{A}$ *violating* if it triggers the event $G$, meaning that for some index $i \in [u]$ this query is:

- either $\textsc{OutDel}(i, C_i, *)$, where there is no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$ so far;

– or HPROVE$(i, C_i, x)$, where there is no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$ so far.

Let us also consider a dynamic "delegation graph" $M = (V, E)$ consisting of all the delegation edges of the form $(i, C_i) \to (j, C_j)$. (Namely, we only look at $u$ special ciphertexts $C_i$ and ignore the rest.) The edge set $E$ of this graph starts empty, but eventually could grow when $\mathcal{A}$ makes honest delegation query HDEL$(i, C_i, j)$ (which returns $C_j$). Moreover, without loss of generality, we assume $M$ is acyclic, as $\mathcal{A}$ get no information by completing the cycle in this graph (i.e., we can simply remove all such edges creating cycles, as $\mathcal{A}$ already knows the answer).

Let $Q$ be the *first* violating query of $\mathcal{A}$, and $j \in [u]$ be the corresponding index of this query. By assumption that $G$ is triggered, $Q$ and $j$ are well defined. Moreover, if $E$ is the current edge set of the delegation graph $M$, we know that there is no delegation path $(1, C_1) \rightsquigarrow (j, C_j)$ in $E$. However, there could potentially be incoming edges from some $(i, C_i) \to (j, C_j)$ in $E$, as long as there is no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$. Going backward from $(j, C_j)$, though, since we know that $M$ is acyclic, we must reach some "source" $(i, C_i)$ where $i > 1$, which has no incoming edges at all (and, hence, no path from $(1, C_1)$ still). Let $(i, C_i)$ be such "source node", which could be the original $(j, C_j)$ in the special case where no delegation edges entered $(j, C_j)$. In either case, however, we know that the query $Q^*$ corresponding to the first time ciphertext $C_i$ appeared in either HDEL$(i, C_i, *)$, HPROVE$(i, C_i, x)$ or OUTDEL$(i, C_i, *)$ had the property that no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$ existed.

In other words, there must exist a query $Q^*$ of $\mathcal{A}$ and an index $i > 1$ such that:

– $Q^*$ is either HDEL$(i, C_i, *)$, HPROVE$(i, C_i, x)$ or OUTDEL$(i, C_i, *)$, and no prior calls of this form where made so far (i.e., ciphertext $C_i$ was not "declared" by $\mathcal{A}$ before $Q^*$).

– At the time $Q^*$ is made, there is no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$.

Let us say that such query $Q^*$ is *$i$-incriminating*. Note, from our definition, each index $i > 1$ could have either zero or one $i$-incriminating query. To summarize,

*event* $G$ $\quad\Longrightarrow\quad$ *there exists* $1 < i \leq u$ *having (unique)* $i$*-incriminating query* $Q^*$.

USING INCRIMINATING QUERY. Recall, in our proof of **Bi-$-Core** security of DEVRF$_2$ under the original legality condition we constructed an attacker $\mathcal{B}^{\mathcal{O}_a(\cdot)}(g, A, B, R, g_1)$ for iBDDH which could simulate all queries of $\mathcal{A}$ except HPROVE$(j, C_j, x)$ and OUTDEL$(j, C_j, *)$, which were prohibited under the original legality condition of $\mathcal{A}$.

We will now construct a different iBDDH attacker $\overline{\mathcal{B}}^{\mathcal{O}_a(\cdot)}(g, A, B, R, g_1)$ which will instead use the fact that $\mathcal{A}$ must make an $i$-incriminating query for some $1 < i \leq u$. $\overline{\mathcal{B}}$ will pick a random index $i \in [2, \ldots u]$, hoping that this is the index corresponding to the $i$-incriminating query $Q^*$. Assuming this guess is correct, we know that $Q^*$ appears before (or exactly at) the first violating query of $\mathcal{A}$, which means that we could have used (but we won't!) the original reduction $\mathcal{B}$ to simulate all the queries of $\mathcal{A}$ before $Q^*$, as none of these queries will have the form HPROVE$(j, C_j, x)$ or OUTDEL$(j, C_j, *)$.

More precisely, $\overline{\mathcal{B}}$ will proceed nearly identically to the original reduction $\mathcal{B}$, but with the following modifications.

1. **Same Simulation.** Initialization of $\mathcal{A}$, challenge ciphertext $C_1$, challenge output $y$, and oracles $H, H'$, HPROVE, and OUTDEL are identical, except when OUTDEL or HPROVE query is $i$-incriminating, as explained below.

2. **Registration Oracle Reg.** Previous attacker $\mathcal{B}$ set all value $A_j = A_1^{\alpha_j}$ for $2 \leq j \leq u$, implicitly setting $a_j = \alpha_j a$. $\overline{\mathcal{B}}$ will do the same for all $j \neq i$. However, for the $i$-th secret key $\overline{\mathcal{B}}$ will chose a random key $a_i \in_r \mathbb{Z}_p$ and honestly set $A_i = g^{a_i}$. In other words, $\overline{\mathcal{B}}$ will actually know the $i$-th secret key.

3. **$i$-Incriminating Query $Q^*$.** Recall, such queries are HDEL$(i, C_i, *)$, HPROVE$(i, C_i, x)$ or OUTDEL$(i, C_i, *)$. Notice, $\overline{\mathcal{B}}$ can test if a ciphertext $C = (A', R', D')$ is equal to $C_i$, by checking that $(A', R') = (A_1, R_1)$ and $e(A_1, R_1) = e(A_i, D')$. Thus, $\overline{\mathcal{B}}$ can indeed test that the query $Q^*$ is $i$-incriminating.
   In this case $\overline{\mathcal{B}}$ knows that the value $D_i$ inside the ciphertext $C_i$ is equal to $D_i = R_1^{a_1/a_i}$. Since $R_1 = R$, $a_1 = a$ and $\overline{\mathcal{B}}$ knows $a_i$, $\overline{\mathcal{B}}$ can compute $D_i^{a_i} = R^a$, and then test if

$$e(D_i^{a_i}, B) \stackrel{?}{=} g_1$$

   In either case, $\overline{\mathcal{B}}$ will abort the entire simulation and output guess $b' = 0$ if and only if the test above passes.

   To explain $\overline{\mathcal{B}}$'s behavior, notice that $e(D_i^{a_i}, B) = e(R^a, g^b) = e(g, g)^{abr}$. Hence, if $g_1 = e(g, g)^{abr}$ the test always passes, and if $g_1$ is random, it almost never passes.

4. **Stuck/Complete Simulation.** When $\overline{\mathcal{B}}$ guesses the "incriminating index" $i$ correctly, we know that $\overline{\mathcal{B}}$ will encounter the incriminating query $Q^*$, and hence output the guess $b'$, before it encounters any of the HPROVE$(j, C_j, x)$ or OUTDEL$(j, C_j, *)$ queries that it cannot simulate. However, when $\overline{\mathcal{B}}$'s guess for $i$ is wrong, we could encounter such a query, or perhaps run $\mathcal{A}$ to completion (say, when $G$ does not happen). In this case, $\overline{\mathcal{B}}$ will output a random guess $b'$.

5. **"IN" Delegation Oracle InDel.** The oracle INDEL$(a', C', j)$ is identical for $j \neq i$ to what was done before by $\mathcal{B}$. Namely, if valid, it simply returns $\mathcal{O}_a((D')^{a^*/\alpha_i})$, where $C' = (A', R', D', \sigma')$. For $j = i$, we know the secret key $a_i$, so we can simply evaluate INDEL honestly using $a'$ and $a_i$.

6. **Honest Delegation Oracle HDel.** When $\mathcal{A}$ calls HDEL$(j_1, (A', R', D', \sigma'), j_2)$, $\overline{\mathcal{B}}$ will first do the delegation and signature checks, rejecting if they fail. Otherwise, if $i \notin \{j_1, j_2\}$, $\overline{\mathcal{B}}$ does the same thing as $\mathcal{B}$, returning $(D')^{a_i/a_j} = (D')^{\alpha_i/\alpha_j}$. Otherwise, either $j_1 = i$ or $j_2 = i$. We treat them differently:

   - If $\mathcal{A}$ calls HDEL$(i, (A', R', D', \sigma'), j)$ where $j \neq i$, $\overline{\mathcal{B}}$ first checks if $(A', R', D') = C_i$, which means $(A', R') = (A_1, R_1)$ and $e(A_1, R_1) = e(A_i, D')$. If this is the case, $\overline{\mathcal{B}}$ knows this is an $i$-incriminating query, and will process it, as explained above.
     Otherwise, $\overline{\mathcal{B}}$ is supposed to return the value

$$(D')^{a_i/a_j} = ((D')^{a_i/\alpha_j})^{1/a} = \mathcal{O}_a((D')^{a_i/\alpha_j})$$

     where $\overline{\mathcal{B}}$ knows $a_i$ and $\alpha_j$. Thus, $\overline{\mathcal{B}}$ can simply return $\mathcal{O}_a((D')^{a_i/\alpha_j})$ using its own oracle. To put it differently, we can pretend that this HDEL query is actually an INDEL query with an adversarial key $a_i$.
   - If $\mathcal{A}$ calls HDEL$(j, (A', R', D', \sigma'), i)$ where $j \neq i$, then we know we have not yet reached the $i$-incriminating query (assuming the guess for $i$ is correct, else we don't care). This

means that the ciphertex
$(A', R', D', \sigma') \neq C_j$, or else the answer to this query will be equal to $C_i$, contradicting the fact that $C_i$ has no incoming delegation edges. In this case we notice that even though key $A_i$ is supposed to be honest, *we can pretend that $a_i$ is an adversarial key, and simulate* $\text{HDEL}(j, (A', R', D', \sigma'), i)$ *as if it is a call to* $\text{OUTDEL}(j, (A', R', D', \sigma'), a_i)$. Specifically, $\overline{\mathcal{B}}$ can perform the delegation check
$e(A', R') \stackrel{?}{=} e(A_j, D')$ and the signature check
$e(H'(A', R'), R') \stackrel{?}{=} e(\sigma', g)$ himself (by evaluating $H'$ according to the standard strategy). Moreover, we know that $(A', R') \neq (A_1, R_1)$ in this case, because $(A', R', D', \sigma') \neq C_j$ and the signature/delegation checks worked. Thus, when we set the value $H'(A', R')$ in our simulation of $H'$, we set it to $H'(A', R') = (A')^{\gamma'}$ for some random $\gamma'$ known to $\overline{\mathcal{B}}$. $\overline{\mathcal{B}}$ will combine this $\gamma'$ with the values of signature $\sigma'$ and secret key $a_i$ known to $\overline{\mathcal{B}}$, and return

$$\text{HDEL}(j, (A', R', D', \sigma'), i) := (\sigma')^{(a_i \gamma')^{-1} \bmod p}$$

The proof of correctness is the same as before, and is omitted.

ANALYSIS OF REDUCTION. The analysis of the reduction is partitioned as to whether $\overline{\mathcal{B}}$ managed to reach the $i$-incriminating query $Q^*$ before being stuck with the simulation. Let us call this event $I$, and notice that it happens at least when $G$ happens and $\mathcal{A}$'s guess $i$ correctly, so

$$\Pr[I] \geq \frac{\Pr[G]}{u-1} = \frac{\Pr[G]}{\text{poly}(k)}$$

When $I$ happens, $\overline{\mathcal{B}}$'s advantage is at least $1 - 1/p = 1 - \text{negl}(k)$. Otherwise, $\overline{\mathcal{B}}$ simply outputs a random $b'$, achieving advantage $1/2$. This gives overall advantage of $\overline{\mathcal{B}}$ equal to

$$\Pr[b' = b] \geq \Pr[I] \cdot (1 - \text{negl}(k)) + (1 - \Pr[I]) \cdot \frac{1}{2} = \frac{1}{2} + \frac{\Pr[G]}{\text{poly}(k)}$$

And since we assume that the iBDDH assumption is true, we know $\overline{\mathcal{B}}$'s advantage must at most $1/2 + \text{negl}(k)$, which means $\Pr[G] = \text{negl}(k)$, completing the proof of Lemma 22. □

## C.7 Proof of One-Time Delegation Security

**Theorem 14.** *The* one-time *delegatable* $\text{DEVRF}_3$ *above satisfies the* ***Bi-$-Core*** *property under the* eBDDH *assumption in the random oracle model. It satisfies the strongest possible legality condition for the attacker (see Definition 21).*

The proof of 1-time delegation is largely similar to the proof of Theorem C.6. As in that proof, we start with the simpler setting of standard legality condition on $\mathcal{A}$, from Definition 10, and then generalize it to the stronger legality in Definition 21.

ORIGINAL LEGALITY CONDITION. Recall, our goal is to construction a reduction $\mathcal{B}$ which uses the attacker $\mathcal{A}$, except must $\mathcal{B}$ must solve the harder eBDDH problem, as opposed to iBDDH. Namely, $\mathcal{B}$ is given $(g, W, A, B, R, g_1)$, where $W = g^{1/a}$, as is no longer given full inversion oracle $\mathcal{O}_a(\cdot)$. Recall also that, in the proof of Theorem 13, the only place where $\mathcal{B}$ used the inversion oracle was to simulate $\text{INDEL}(a', C', i)$ oracle.

Our main idea is to use the fact that in the 1-time delegation scenario, each valid ciphertext $C' = (A', R', D', \sigma')$ submitted to OUTDEL must have $A' \in \{A_1, \ldots, A_u\}$, while each such ciphertext submitted to INDEL must have $A' \notin \{A_1, \ldots, A_u\}$ (or else the attacker breaks the discrete log of some unknown honest key $A_i$). Thus, if previously we only used the signature $\sigma$ to help us simulate OUTDEL queries, by effectively extracting the value $\mathsf{DH}(A', R')$ for the submitted ciphertext $C'$, now we can use it instead for helping simulate INDEL queries as well, since those queries operate on *disjoint sets of public keys* $A$.

More concretely, our new attacker $\mathcal{B}(g, W, A, B, R, g_1)$ for eBDDH will operate identically with the previous attacker in Theorem C.6 (which used the inversion-oracle), except with the following changes:

1. Initialization of $\mathcal{A}$, challenge ciphertext $C_1$, challenge output $y$, and oracles REG, $H$, HPROVE, and HDEL are identical.

2. **Hash Queries to $H'$.** Recall, our previous attacker set $H'(A_1, R_1) = g^{\gamma_1}$ for random $\gamma_1 \in_r Z_p$, but all other queries $H'(A', R') = (A')^{\gamma'}$ for fresh random $\gamma' \in_r Z_p$. Now, we still set $H'(A_1, R_1) = g^{\gamma_1}$, but set other queries $(A', R') \neq (A_1, R_1)$ as follows:

   - If $A' \in \{A_1, \ldots, A_u\}$, set $H'(A', R') = (A')^{\gamma'}$ for fresh random $\gamma' \in_r Z_p$, as before.
   - Otherwise, set $H'(A', R') = W^{\gamma}$ for fresh random $\gamma \in_r Z_p$, where $W = g^{1/a}$ comes from the input of $\mathcal{B}$.

3. **"OUT" Delegation Oracle OutDel.** When $\mathcal{A}$ calls
   $\text{OUTDEL}(i, (A', R', D', \sigma'), a^*)$, for any secret key $a^*$, any such (allowed) query must have $A' = A_i$, by our stricter delegation check. Hence, the oracle $H'(A', R') = (A')^{\gamma'}$, as in the proof of Theorem C.6 (this uses the fact $(A', R') \neq (A_1, R_1)$, as this query is not allowed by legality of $\mathcal{A}$, even if $i = 1$.) Hence, we can use the same strategy as before. Namely, the value $\mathcal{B}$ can return the value

   $$\text{OUTDEL}(i, (A', R', D', \sigma'), a^*) := (\sigma')^{(a^*\gamma')^{-1} \bmod p}$$

   as before, as this value is equal to $(D')^{a_i/a^*}$. The proof of this is the same as in Theorem 12.

   c

4. **"IN" Delegation Oracle InDel.** When $\mathcal{A}$ calls
   $\text{INDEL}(a', (A', R', D', \sigma'), i)$, for any secret key $a'$, we first check if $a' \in \{a_1, \ldots, a_u\}$, by checking that $g^{a'} \stackrel{?}{=} A_i$ for all $i$. If this is the case, $\mathcal{B}$ can compute $a = a'/\alpha_i$, which trivially allows it to win its game by checking whether $g_1 \stackrel{?}{=} e(B, R)^a$.

   Otherwise, we know that $H'(A', R') = W^{\gamma}$ for some random $\gamma$. From delegation check, we also know that $A' = g^{a'}$ and $R' = D'$, and from the signature check we know that $e(H(A', R'), R') = e(\sigma', g)$. This means

   $$e(\sigma', g) = e(H(A', R'), R') = e(W^{\gamma}, R') = e((R')^{\gamma/a}, g)$$

   This implies that $(R')^{1/a} = (\sigma')^{1/\gamma}$. But $\mathcal{A}$ expects to see

   $$(D')^{a'/a_i} = (R')^{a'/(\alpha_i a)} = ((R')^{1/a})^{a'/\alpha_i} = ((\sigma')^{1/\gamma})^{a'/\alpha_i} = (\sigma')^{a'/(\gamma\alpha_i)}$$

Thus, $\mathcal{B}$ can complete the simulation by responding with

$$\textsc{InDel}(a', (A', R', D', \sigma'), i) = (\sigma')^{a'/(\gamma\alpha_i)}$$

This completes our reduction for the basic legality condition.

Stronger Legality Condition. Finally, we show how to extend our proof to the optimal legality condition given in Definition 21. This will be done very similar to the proof of Theorem 13, but slightly simpler due to the more limited structure of the delegation graph $M$ used in the proof of Theorem 13.

In particular, recall "special" ciphertexts $C_i = (A_1, R_1, D_i, \sigma)$, where $D_i = R_1^{a_i/a_1}$ and $\sigma = H'(A_1, R_1)^{r_1}$. Except for $i = 1$, none of these ciphertexts can be delegated. Thus, the "gap" event $G$ between the 2 legality conditions of $\mathcal{A}$ becomes simply:

- $\mathcal{A}$ made a call to $\textsc{HProve}(i, C_i, x)$ for some $i > 1$, where $x$ is the challenge input produced by $\mathcal{A}_1$, but did not query $\textsc{HDel}(1, C_1, i)$.

If $G$ does not happen, this corresponds to the original legality condition, for which we already finished the proof. Thus, it remains to show that for any PPT attacker $\mathcal{A}$, we have $\Pr[G] = \text{negl}(k)$. Once again, this is proven similarly to the corresponding proof of Lemma 22 in Section C.6.

In particular, we will build a reduction $\overline{\mathcal{B}}$ to $\mathsf{eBDDH}$ from any attacker $\mathcal{A}$ which triggered the gap event $G$. Since our 1-time delegatable scheme, $\text{DEVRF}_3$ is really a special case of the general scheme $\text{DEVRF}_2$ considered in Section C.6, our reduction $\overline{\mathcal{B}}$ can be identical to the "old" reduction — call it $\overline{\mathcal{B}}'$ — used in the proof of Lemma 22, except we need to make sure we no longer need to use the inversion oracle $\mathcal{O}_a(\cdot)$, and only use the value $W = g^{1/a}$. Fortunately, we can do it using the same technique as in the proof of the original legality condition, by basically choosing a random index $i > 1$ (hoping it corresponds to the incriminating query of $\mathcal{A}$), and effectively treating the known key $a_i$ as adversarial. We highlight the differences here:

1. **Registration Oracle Reg.** We simulate exactly like the previous reduction $\overline{\mathcal{B}}'$. We choose a random index $i > 1$ at random, and set all public keys $A_j = A_1^{\alpha_j}$ for $j \neq i$ for random $\alpha_j$. However, for the $i$-th secret key $\overline{\mathcal{B}}$ will chose a random key $a_i \in_r \mathbb{Z}_p$ and honestly set $A_i = g^{a_i}$. In other words, $\overline{\mathcal{B}}$ will actually know the $i$-th secret key.

2. **New Simulation of $H'$.** As done earlier in the section, we set $H'(A', R')$ as follows:

   - If $(A', R') = (A_1, R_1)$, set $H'(A_1, R_1) = g^{\gamma_1}$, for random $\gamma_1 \in_r Z_p$
   - If $A' \in \{A_j \mid j \neq i\}$, set $H'(A', R') = (A')^{\gamma'}$ for fresh random $\gamma' \in_r Z_p$.
   - Otherwise, set $H'(A', R') = W^{\gamma}$ for fresh random $\gamma \in_r Z_p$.
     (Note, this critically includes the values $H'(A_i, R')$.)

   Due to the 1-time delegation nature of $\text{DEVRF}_3$, this restriction on $H'$ still allows us to correctly simulate all $\textsc{OutDel}(j, C', *)$ calls for $j \neq i$, as they will use $H'(A_j, R') = (A_j)^{\gamma'}$. While for $j = i$ we can simply use the secret key $a_i$.

3. **Places Previous $\overline{\mathcal{B}}'$ used $\mathcal{O}_a(\cdot)$.** Examining the proof of Lemma 22, there were only two places where $\overline{\mathcal{B}}'$ used the inversion oracle $\mathcal{O}_a(\cdot)$:

– **Case 1:** When $\mathcal{A}$ called $\textsc{InDel}(a', (A', R', D', \sigma'), j)$. For $j = i$, our new reduction $\overline{\mathcal{B}}$ can simply use $a_i$, as did $\overline{\mathcal{B}}$. However, for $j \neq i$, instead of calling (no longer present) oracle $\mathcal{O}_a((D')^{a^*/\alpha_j})$, we use the same strategy we used at the beginning of this section to extract the answer from the signature $\sigma'$ supplied by the attacker. Namely, we know that $g^{a'} = A'$, $R' = D'$, $H(A', R') = W^\gamma$ in our simulation (as otherwise $a' = a_j = \alpha_j a$, for $j \neq i$, which allows us to break $\mathsf{eBDDH}$ trivially), and $e(H(A', R'), R') = e(\sigma', g)$. This means

$$e(\sigma', g) = e(H(A', R'), R') = e(W^\gamma, R') = e((R')^{\gamma/a}, g)$$

This implies that $(R')^{1/a} = (\sigma')^{1/\gamma}$. But $\mathcal{A}$ expects to see

$$\begin{aligned}(D')^{a'/a_j} &= (R')^{a'/(\alpha_j a)} \\ &= ((R')^{1/a})^{a'/\alpha_j} \\ &= ((\sigma')^{1/\gamma})^{a'/\alpha_j} = (\sigma')^{a'/(\gamma\alpha_j)}\end{aligned}$$

Thus, $\overline{\mathcal{B}}$ can complete the simulation by responding with

$$\textsc{InDel}(a', (A', R', D', \sigma'), j) = (\sigma')^{a'/(\gamma\alpha_j)}$$

– **Case 2:** When $\mathcal{A}$ called $\textsc{HDel}(i, (A', R', D', \sigma'), j)$ for $j \neq i$. However, by 1-time delegation check we know that $A' = A_i$, which means $H(A_i, R') = W^\gamma$ as well. And hence we can use exactly the same strategy as in the previous **Case 1**, effectively treating the known $a_i$ as the adversarial key and returning

$$\textsc{HDel}(i, (A', R', D', \sigma'), j) = (\sigma')^{a_i/(\gamma\alpha_j)}$$

This completes the new reduction $\overline{\mathcal{B}}$ to the $\mathsf{eBDDH}$ problem, and the overall proof of the stronger legality condition of $\text{DEVRF}_3$. □