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

Erik Aronesty[1], David Cash[2], Yevgeniy Dodis[3*], Daniel H. Gallancy[1], Christopher Higley[1], Harish Karthikeyan[3], and Oren Tysor[1]

[1] Atakama
[2] University of Chicago
[3] New York University

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 a real-world company.

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*.

* Partially supported by gifts from VMware Labs and Google, and NSF grants 1619158, 1319051, 1314568.

Alice works on the Desktop and periodically generates large documents D_1, $D_2 \ldots$, that she might want to index separately.[4] 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.

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.

[4] 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.

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 and can be found in the full version of the paper [3]. 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 the full version of the paper [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 the full version of the paper [3], 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) [35]. 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, 34, 35], VRFs are quite efficient in the random oracle model. In particular, several such efficient constructions are given the CFRG VRF standard [29, 30].

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*[5] — 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 the full version [3].). 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 [26].[6] 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*[7] threshold CCA decryption with keys residing in $\mathbb{Z}_p$. Both options seem challenging.

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

[6] As other prior distributed VRFs were either interactive [23, 33], or had no verifiability [2, 36] or offered no formal model/analysis [16, 17, 21, 32, 41].

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

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 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 2. 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 2. Moreover, this construction is very friendly to all our extensions. If the EVRF is a threshold (resp. delegatable) — see Definitions 3,4, — 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 1. 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 6.2 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 [42], Feldman VSS [25], 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 7.2,7.3,7.4 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 7.2 (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 7.3 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 7.3,7.4).

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, 28] and *Public-Key Encrypted Keyword Search* (PEKS) [1, 4, 8, 10, 40, 44]. 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 $D_1, D_2, \ldots$.[8]

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.[9] 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.

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.

[8] 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.

[9] Unfortunately, as surveyed by Cash *et al.* [14] and further studied by [20, 31] (and others), *all* SSE schemes in the literature do not achieve the strongest possible keyword privacy and suffer from various forms of information leakage.

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.

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!)

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 machines (PPTs). We use poly(k) to denote a polynomial function, and 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 the full version of our paper [3]. 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, 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 also consider extensions to this primitive, adding features such as distribution, delegation, and update. Due to space constraints, we defer the discussions to the full version of the paper [3].

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, poly(k) to denote a polynomial function, and 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* ESI = (KGen, Prep, Index, S-Split, S-Core, Finalize) *such that:*

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

We require the following security properties from this primitive:

1. ***Correctness**: with prob.* 1 *(resp.* $(1 - negl(k))$*) over randomness of* $\textsc{KGen}$ *and* $\textsc{Prep}$*, 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* $\textsc{S-Prove}(PK, SK, c^*, w)$ *with* $w \in (D_1 \backslash D_2) \cup (D_2 \backslash D_1)$ *and* $|D_1| = |D_2|$*, for variables* SK, c^*, D_1, D_2, w *defined below:*
$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} (PK, SK) \leftarrow \textsc{KGen}(1^k); \\ (D_1, D_2, st) \leftarrow \mathcal{A}_1^{\textsc{S-Prove}(PK, SK, \cdot, \cdot)}(PK); \\ b \leftarrow \{0, 1\}; \\ (E^*, c^*) \leftarrow \textsc{BldIdx}(PK, D_b); \\ b' \leftarrow \mathcal{A}_2^{\textsc{S-Prove}(PK, SK, \cdot, \cdot)}(E^*, c^*, st) \end{array} \right] \leq \frac{1}{2} + negl(k)$$
4. ***Privacy-Preserving** [10]: 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:*
$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} (PK, SK) \leftarrow \textsc{KGen}(1^k); \\ (D_1, D_2, st) \leftarrow \mathcal{A}_1(PK, SK); \\ b \leftarrow \{0, 1\}; \\ (E^*, c^*) \leftarrow \textsc{BldIdx}(PK, D_b); \\ b' \leftarrow \mathcal{A}_2(c^*, st) \end{array} \right] \leq \frac{1}{2} + negl(k)$$

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.

[10] 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 $\textsc{Prep}$ 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 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 the full version of the paper [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 the full version of the paper [3].

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 the full version of the paper [3].

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 4.2, 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 2. *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:*

- $\textsc{Gen}(1^k) \rightarrow (PK, SK)$*: outputs the public/secret key pair.*
- $\textsc{Encap}(PK) \rightarrow (C, T)$*: 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.*[11]
- $\textsc{Core}(SK, R', x) = z'$*: 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* $\textsc{Gen}$ *and* $\textsc{Encap}$*, 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*

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

[11] The algorithm $\textsc{Split}$ is not technically needed, as one can always set $R = C$. In fact, this will be the case for our EVRF in section 5.1. However, one could envision EVRF constructions where the $\textsc{Split}$ procedure can do a non-trivial (input-independent) part of the overall $\textsc{Prove} = \textsc{Core}(\textsc{Split})$ 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 7.1.

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* Prove$(PK, SK, \cdot, \cdot)$*, for variables SK, C, x defined below, the following holds:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} (PK, SK) \leftarrow \textsc{Gen}(1^k); \\ (C, T) \leftarrow \textsc{Encap}(PK); \\ (x, st) \leftarrow \mathcal{A}_1^{\textsc{Prove}(PK,SK,\cdot,\cdot)}(PK, C); \\ y_0 = \textsc{Comp}(T, x); \quad y_1 \leftarrow \{0,1\}^{|y_0|}; \\ b \leftarrow \{0,1\}; \quad b' \leftarrow \mathcal{A}_2^{\textsc{Prove}(PK,SK,\cdot,\cdot)}(y_b, st) \end{array} \right] \le \frac{1}{2} + negl(k)$$

We present a construction of our EVRF in section 5.1.

Remark 4. We note that any valid ciphertext C implicitly defines a standard verifiable random function (VRF). In particular, the value $z = \textsc{Prove}(SK, C, x)$ could be viewed as the VRF proof, which is accepted iff $\textsc{Post}(PK, z, C, x) \neq \perp$.

Remark 5. We reiterate that our pseudorandomness definition does not give the attacker "unguarded" access to the Core procedure, but only "Split-guarded" access to Prove = Core(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 Extensions to EVRFs

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 6.1.

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.,

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

where $\textsc{Eval}(PK, SK, C, x) = \textsc{Post}(PK, \textsc{Prove}(SK, c, x), C, x)$. The formal discussion about the syntax and security of this primitive can be found in Section 7.1.

Protocol Standard EVRF

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

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

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

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

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

$\textsc{Post}(PK, z, C', x)$
    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 1. Standard EVRF $=$ ($\textsc{Gen}, \textsc{Encap}, \textsc{Comp}, \textsc{Split}, \textsc{Core}, \textsc{Post}$).

5 Our constructions

We begin by presenting the standard EVRF construction in Section 5.1. We then present a generic construction of our ESI in Section 5.2.

5.1 Standard EVRF

We now present the standard EVRF construction, presented in Construction 1.

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

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

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 $\textsc{Post}(A, \textsc{Core}(a, \textsc{Split}(A, R), x), R, x)$ which concludes the proof.

To prove **Uniqueness**, consider any tuple $(PK = A, C = R, x, z_1, z_2)$. Further, let $y_1 = \textsc{Post}(A, z_1, R, x)$ and $y_2 = \textsc{Post}(A, z_2, R, x)$. 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 can prove the following result in the full version of the paper [3].

Theorem 1. *The standard* EVRF *given in Construction 1 satisfies the **\$-Core** property under the* BDDH *assumption in the random oracle model.*

5.2 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

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 $y_w$ = 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 2. Generic ESI = (KGen, Prep, Index, S-Split, S-Core, Finalize).

$w \in D$ 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, 37, 38] or cuckoo hash [39], 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 DS = (Construct, Find) is any data structure supporting the following two operations:

- Construct(D) $\to E$: outputs the index E on an input document D.
- 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 DS = (Construct, Find) and and EVRF = (Gen, Encap, Comp, Split, Core, Post). This composition is given below in Construction 2.

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.

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 the full version of the paper [3] we prove the following theorem:

Theorem 2. *If* EVRF *satisfies the* ***\$-Core*** *property, then Encapsulated Search Index is* ***CCA*** *secure. Further, if the* EVRF *(resp.* DS *) is threshold and/or delegatable, the resulting ESI inherits the same.*

6 Threshold Encapsulated Verifiable Random Functions

In this section, we formally introduce the primitive known as a Threshold EVRF in Section 6.1. We then present a construction of Threshold EVRF in Section 6.2 but defer the security proof due to space constraints. The proof can be found in the full version of the paper [3].

6.1 Definition of Threshold (or Distributed) EVRFs

Definition 3. *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:*

- $\textsc{Gen}(1^k,t,n) \to (PK, \boldsymbol{SK} = (sk_1,\dots,sk_n), \boldsymbol{VK} = (vk_1,\dots,vk_n))$*: 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* 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*.*
- $\textsc{Split}(PK,n,C') = (R'_1,\dots R'_n)$*: outputs* n *handles* $R'_1,\dots,R'_n$ *from full ciphertext* C'*.*
- $\textsc{D-Core}(sk_i,R'_i,x) = z'_i$*: 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.*
- $\textsc{Combine}(PK,C',z'_{i_1},\dots,z'_{i_t},x) = y'$*: uses the partial evaluations* $z'_{i_1},\dots,z'_{i_t}$ *to compute the final value of EVRF on input* x*.*[12]

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

- $\textsc{Prove}(\mathbf{SK},i,C,x) = \textsc{D-Core}(sk_i,R_i,x)$*, where* $(R_1,\dots,R_n) = \textsc{Split}(PK,n,C)$.
- $\textsc{Eval}(\mathbf{SK},i_1,\dots,i_t,C,x)$*: For* $j = 1\dots t$*, 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* $\textsc{Combine}(PK,C,z_{i_1},\dots,z_{i_t},x)$*.*

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* Gen *and* Encap*, for honestly generated ciphertext* C*:* $\textsc{Eval}(\mathbf{SK},i_1,\dots,i_t,C,x) = \textsc{Comp}(T,x)$

[12] 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 Combine). See also the definition of Eval below to explicitly model this assumption.

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,

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

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,*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \{i_1, \ldots, i_{t-1}, st\} \leftarrow \mathcal{A}_0(1^k, t, n); \\ (PK, \mathbf{SK}, \mathbf{VK}) \leftarrow \textsc{Gen}(1^k, t, n); \\ (C, T) \leftarrow \textsc{Encap}(PK); \\ (R_1, \ldots, R_n) = \textsc{Split}(PK, n, C); \\ (x, st) \leftarrow \mathcal{A}_1^{\textsc{Prove}(\mathbf{SK}, \cdot, \cdot, \cdot)}(PK, C, \mathbf{VK}, \mathbf{SK}', st) \\ y_0 = \textsc{Comp}(T, x); \quad y_1 \leftarrow \{0,1\}^{|y_0|}; \\ b \leftarrow \{0,1\}; \quad b' \leftarrow \mathcal{A}_2^{\textsc{Prove}(SK, \cdot, \cdot)}(y_b, st) \end{array} \right] \leq \frac{1}{2} + negl(k)$$

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

We present a construction of our threshold EVRF in section 6.2.

Remark 6. 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 $\textsc{Gen-Vfy}$ 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 6.2, can easily achieve efficient distributed key generation using techniques of Gennaro *et al.* [27].

Remark 7. Note that when $t = n = 1$, our threshold EVRF implies the the standard EVRF definition (Definition 2), where $\textsc{Post}$ algorithm first runs $\textsc{Shr-Vfy}$ on the single share z and then, if successful, runs $\textsc{Combine}$ 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., $\textsc{Shr-Vfy}$'ed) shares z_i.

6.2 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 1 with the ideas of Shamir's Secret Sharing [42], Feldman VSS [25], and the fact that the correctness of all computations is easily verified using the pairing.

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).

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

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

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

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

- 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 $f(0) = \sum_{\ell=1}^{t} \lambda_\ell \cdot f(i_\ell)$ 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 $\ell = 1, \dots, t$, $\textsc{Shr-Vfy}(PK, vk_{i_\ell}, z_{i_\ell}, x) = 1$ and $\textsc{Shr-Vfy}(PK, vk_{j_\ell}, z_{j_\ell}, x) = 1$. Therefore, we have that for $\ell = 1, \dots, t$:

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

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

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

From Equation (3):

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

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 the full version of the paper [3].

Theorem 3. *If Construction 1 satisfies the **\$-Core** property of standard* EVRF*, then Construction 3 satisfies the **\$-DCore** property of threshold* EVRF*. By Theorem 1, it follows that Construction 3 satisfies the **\$-DCore** property under the* BDDH *assumption in the random oracle model.*

7 Delegatable Encapsulated Verifiable Random Functions

In this section, we formally introduce the primitive known as a Delegatable EVRF in Section 7.1. This definition captures different levels of delegatability and we present constructions that satisfy these levels in Sections 7.2,7.3, and 7.4. The security proofs are deferred to the appendix.

7.1 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 9.

We define three levels of pseudorandomness security for delegatable EVRFs.

Definition 4. *An EVRF* $= (\textsc{Gen}, \textsc{Encap}, \textsc{Comp}, \textsc{Split}, \textsc{Core}, \textsc{Post})$ *is* delegatable *if there exists polynomial-time procedures* $\textsc{Del}$ *and* $\textsc{Same}$*, such that:*

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

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

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

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*,*

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

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

$$\textsc{Same}(PK_1, C_1, PK_2, C_2) = 1 \implies \\ \forall x \ \textsc{Eval}(SK_1, C_1, x) = \textsc{Eval}(SK_2, C_2, x)$$

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:*

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

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

(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****.*

Now, we can define the oracles. As alluded to earlier, there are significant subtleties in both the syntax and security of such a primitive. We defer this exposition to the appendix for want of space. This discussion can be found in the full version of the paper [3].

To adequately capture these 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 $1 \leq i \leq q$ 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 $1 \leq i, j \leq q$ 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 $1 \leq i \leq q$ 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 $C' = \text{Del}(SK_i, C, SK/PK)$.
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.

Remark 8. **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 9. 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 the full version of the paper [3], 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.

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).

Remark 10. 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 11. 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.

7.2 Construction of Basic Delegatable EVRF

We now show that our original EVRF Construction 1 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 5.1, 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 $e(A', R') \overset{?}{=} e(A, D')$ 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 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 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 $e(A, R) \overset{?}{=} e(A_1, D_1)$ and $e(A', R') \overset{?}{=} e(A_2, D_2)$ 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: 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 to prove the following:

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

The proof of the above theorem is deferred to full version of paper [3].

Delegation Attack on Stronger Legality. We briefly mentioned in Section 7.1 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 $\text{H}\textsc{Prove}(2, C_2, x)$ (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 $\text{H}\textsc{Prove}(2, C_2, x)$ and break the scheme.

7.3 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] σ 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 σ 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 DEVRF_2 is essentially the same as DEVRF_1, its correctness follows the same argument. In particular, we notice that the original signature σ 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, $e(H'(A',R'),R') \overset{?}{=} e(\sigma',g)$, is important to ensure that the value σ' is information-theoretically determined from the value (A',R'), so it is fine to not include σ under H.

Protocol 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 $R = D = g^r, S = A^r, \sigma = H'(A, R)^r$.
  **return** ciphertext $C = (A, R, D, \sigma)$ 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 $SK_1 = a_1, SK_2 = a_2, C_1 = (A, R, D_1, \sigma)$.
  **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** $C_2 = (A, R, D_2)$.

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** $(A', R')$.

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

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** $y'$.

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

Also, since the delegation procedure Del simply copies the values A, R and σ, and only modifies the value D_1, the **Delegation-Completeness** and **Delegation-Soundness** of DEVRF_2 holds as it did for 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 the full version of the paper [3], we also show how the addition of the "BLS signature" σ and the new signature check allow us to prove the following theorem:

Theorem 5. *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 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 the full version of the paper [3], where we also show the following result:

Theorem 6. *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 [3]).*

7.4 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" ($e(A,R) \overset{?}{=} e(A_1, D_1)$) by a stricter "equality check"($(A,R) \overset{?}{=} (A_1, D_1)$) which means that the ciphertext C_1 was directly created for public key $A_1 = A$. We call the resulting 1-time-delegatable construction DEVRF_3. In the full version of the paper [3] we show that DEVRF_3 satisfies bidirectional-delegation security, but now under a much weaker (non-interactive) eBDDH assumption:

Theorem 7. *The* one-time *delegatable* 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 [3]).*

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

8 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 2, and by using any updatable sub-linear DS with an appropriate (delegatable and/or threshold) EVRF from the earlier sections, we get:

Theorem 8. *We have an updatable* ESI *(see [3]) 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* DEVRF_1*)*
- ***Uni CCA secure*** *under* BDDH *assumption (by using* DEVRF_2*)*
- ***Bi CCA secure*** *under* iBDDH *assumption (by using* DEVRF_2*)*
- ***One Time CCA secure*** *under* eBDDH *assumption (by using* DEVRF_3*)*

Commercial Product. This theorem forms the backbone of a commercially available product that has been in the market since 2020. 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.[13] 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 the full version of the paper [3].

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, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 205–222. Springer, Heidelberg, August 2005.
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*, pages 1993–2010. ACM Press, October 2018.
3. Erik Aronesty, David Cash, Yevgeniy Dodis, Daniel H. Gallancy, Christopher Higley, Harish Karthikeyan, and Oren Tysor. Encapsulated search index: Public-key, sub-linear, distributed, and delegatable (full version). `https://cs.nyu.edu/~dodis/ps/esi.pdf`, 2021.

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

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.
7. Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 226–243. Springer, Heidelberg, February 2006.
8. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 506–522. Springer, Heidelberg, May 2004.
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, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 50–67. Springer, Heidelberg, August 2007.
11. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.
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*, pages 320–329. ACM Press, November 2005.
13. Ran Canetti and Shafi Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In Stern [43], 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*, pages 668–679. ACM Press, October 2015.
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*, volume 3531 of *LNCS*, pages 442–455. Springer, Heidelberg, June 2005.
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, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, Heidelberg, August 1998.
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*, pages 79–88. ACM Press, October / November 2006.

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*, volume 2567 of *LNCS*, pages 1–17. Springer, Heidelberg, January 2003.
24. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005.
25. Frank A. Feldman. Fast spectral tests for measuring nonrandomness and the DES. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 243–254. Springer, Heidelberg, August 1988.
26. 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.
27. 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.
28. Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216.
29. Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably preventing DNSSEC zone enumeration. In *NDSS 2015*. The Internet Society, February 2015.
30. 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.
31. Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 965–976. ACM Press, October 2012.
32. Keep. The Keep Random Beacon: An Implementation of a Threshold Relay, 2020. https://docs.keep.network/random-beacon/.
33. 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*, volume 8257 of *LNCS*, pages 251–270. Springer, Heidelberg, November 2013.
34. Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 597–612. Springer, Heidelberg, August 2002.
35. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.
36. Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Stern [43], pages 327–346.
37. Moni Naor and Eylon Yogev. Tight bounds for sliding bloom filters. *Algorithmica*, 73(4):652–672, December 2015.

38. 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.
39. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122 – 144, 2004.
40. 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*, pages 376–379. ACM Press, March 2009.
41. 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.
42. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
43. Jacques Stern, editor. *EUROCRYPT'99*, volume 1592 of *LNCS*. Springer, Heidelberg, May 1999.
44. 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.