Iterative Oblivious Pseudo-Random Functions and Applications

Erik-Oliver Blass[1], Florian Kerschbaum[2], and Travis Mayberry[3]

[1] Airbus
[2] University of Waterloo
[3] United States Naval Academy

Abstract. We consider the problem of a client querying an encrypted binary tree structure, outsourced to an untrusted server. While the server must not learn the contents of the binary tree, we also want to prevent the client from maliciously crafting a query that traverses the tree out-of-order. That is, the client should not be able to retrieve nodes outside one contiguous path from the root to a leaf. Finally, the server should not learn which path the client accesses, but is guaranteed that the access corresponds to one valid path in the tree. This is an extension of protocols such as structured encryption, where it is only guaranteed that the tree's encrypted data remains hidden from the server.

To this end, we initiate the study of Iterative Oblivious Pseudorandom Functions (iOPRFs), new primitives providing two-sided, fully malicious security for these types of applications. We present a first, efficient iOPRF construction secure against both malicious clients and servers in the standard model, based on the DDH assumption. We demonstrate that iOPRFs are useful to implement different interesting applications, including an RFID authentication protocol and a protocol for private evaluation of outsourced decision trees. Finally, we implement and evaluate our full iOPRF construction and show that it is efficient in practice.

1 Introduction

Structured encryption allows a data owner to encrypt data arranged in a data structure and store it at an untrusted server [9]. The crucial property of structured encryption is that the data owner can later compute a special decryption key for the server which permits the server to decrypt and parse a well defined component of the data structure. A typical example for structured encryption is data arranged in a graph, encrypted and outsourced to a server, and the owner computing keys for decryption of sub-graphs. Computation of decryption keys is possible despite the owner retaining only a constant-sized master key. Keyword-searchable encryption is also a special-case of structured encryption where the graph is composed of many linked lists, one for each keyword, containing all the documents that match that keyword.

New Applications In this paper, we introduce a twist to the standard application scenario of structured encryption. A third party, separate from the data owner and server, which we call the client, can ask the data owner for permission to retrieve a specific component of the owner's data structure. The owner is said to *delegate* access to this portion of his data to the client. However, the data owner and client do not trust each other, and the client does not want to reveal to the (potentially) malicious owner which part of the data structure they are interested in. At the same time, the owner wants some guarantee that the client is restricted to a specific component of their data structure and might even put constraints on that component, e.g., where it begins, how many elements it contains, etc.

We focus on tree data structures, but in return offer more powerful confinement control for the data owner than standard structured encryption. In addition to decryption keys enabling decryption of a sub-tree for the client, the data owner can also compute keys which enable the client to access only one path, from the root of the tree to a leaf. Moreover, the client can ask to decrypt a path in an iterative, adaptive fashion instead of querying the owner for the whole path at once. Adaptive queries are necessary to support iterative scenarios where the client will parse the tree node by node, obliviously asking the owner to decrypt the next node in the tree only after fetching and decrypting the previous node. For example, after decrypting one node of a binary tree, the client can obliviously query the owner for the decryption key of either the left or right child, depending on the current node's data content. At the same time, the data owner wants to ensure that the client can only ask to decrypt one single node which is a child of the current node, so that the client is confined to decrypting exactly one path and cannot arbitrarily "jump around" in the data structure.

This new setting of mutually untrusted data owner, server, and client has several interesting real-world applications. While we later focus on two specific applications, one for RFID authentication and one for privacy-preserving decision tree evaluation, we stress that techniques in this paper are general and useful in other scenarios, too. As soon as data is tree-structured (XML data, databases using B+ trees or hash maps, hash trees, search trees, heaps, ...) and should be adaptively parsed, our techniques will be required.

Technical Challenges A straightforward, intuitive approach to providing such adaptive queries might be for the data owner to apply an Oblivious Pseudo-Random Function (OPRF) as the PRF to encrypt nodes. For a tree of height ℓ, owner and client then run ℓ instances of the OPRF such that the client always learns the key for the next node on the path they are interested in, and the owner learns nothing. To actually fetch the next node from the server in an oblivious fashion, the client could employ standard PIR or OT protocols. However, this approach is only secure against semi-honest clients that stick to the rule of asking for the decryption key of one child node of the current node. The difficulty lies in making parsing the tree structure secure against a fully-malicious client without reverting to general, yet expensive techniques like maliciously secure two-party computation and expensive general Zero-Knowledge Proofs.

This Paper Consequently, we introduce the notion of iterative Oblivious Pseudo-Random Functions (iOPRFs) and introduce first candidate constructions. An iOPRF is an ℓ round two party protocol between a sender and a receiver. Its definition captures the intuition that the receiver can adaptively query ℓ input bits x_i in ℓ rounds such that in the end they learn outputs $\mathrm{PRF}_K(x_1), \dots, \mathrm{PRF}_K(x_1 \dots x_\ell)$ for key K chosen by the sender, and the sender learns nothing. If such an iOPRF is used to encrypt the nodes, then fetching a wrong node is useless for the client, as they cannot decrypt it anyways.

Our new candidate iOPRF construction is based on a careful adaptation of the PRF by Naor and Reingold [35]. We first augment the Naor and Reingold PRF to become an iterative Pseudo-Random Function (iPRF) which has the property that, for input strings with the same prefix, its generated output also shares the same prefix. As a warm-up, we then use a similar trick as Freedman et al. [16] to convert the iPRF to an iOPRF. This first iOPRF is OT-based and elegant, yet it only offers one-sided security [21] against a malicious receiver and semi-honest sender. We then present our main construction, an iOPRF which is secure against malicious sender and malicious receiver. We achieve malicious security by using efficient zero-knowledge proofs for DH-based statements over elliptic curves and avoid costly maliciously secure oblivious transfer (OT). We implement and benchmark our new iOPRF construction to show that it is practical and efficient.

In summary, our **main contributions** are:

- The definition of the new cryptographic primitives of iPRF and iOPRF which extends repeated OPRF constructions with security constraints on the client's input.
- A candidate construction which is efficient and provably secure under the Decisional Diffie-Hellman assumption in the standard model. To show its practicality, we implement our construction and evaluate its performance. The implementation is available for download [1].
- The integration of our primitive into several example applications, such as RFID authentication and privacy-preserving decision tree evaluation.

2 Background and Related Work

Before introducing iPRFs, iOPRFs, and their constructions, we briefly revisit seminal PRF and OPRF schemes and some useful security definitions. They will be helpful in understanding the intuition behind iPRFs and iOPRFs.

PRFs and OPRFs While there exist many different PRFs [3, 11, 15, 18, 32, 35] and OPRFs [4, 10, 16, 23, 25, 31], we present the DH-based techniques by Naor and Reingold [35] and Freedman et al. [16], as our constructions are build on their main idea.

Let $\mathbb{G}$ be a group of prime order p where the DDH assumption holds, and g is a random generator of $\mathbb{G}$. For a security parameter λ, we set $|p| = \mathsf{poly}(\lambda)$.

Construction 1 (Naor and Reingold Function)**.** For any $\ell \in \mathbb{N}$, consider function family (ensemble) $F_K(x) : (\mathbb{Z}_p)^{\ell+1} \times \{0,1\}^\ell \to \mathbb{G}$, where key K is defined as sequence $K = (\alpha_0, \ldots, \alpha_\ell)$ of $\ell + 1$ random elements α_i from $\mathbb{Z}_p$. For any ℓ bit input $x = x_1 \ldots x_\ell$, function F_K is defined by

$$F_K(x) = (g^{\alpha_0})^{\prod_{x_i=1} \alpha_i}.$$

Function F_k holds the following important randomness property. We will come back to it later in the proof of our own construction.

Definition 1 (Naor and Reingold Pseudo-Randomness). *For any $\ell \in \mathbb{N}$, function family $F_K(x) : (\mathbb{Z}_p)^{\ell+1} \times \{0,1\}^\ell \to \mathbb{G}$ has* pseudo-random output, *iff for every PPT distinguisher $\mathcal{D}$, there exists a negligible function ϵ such that for sufficiently large λ*

$$|Pr[\mathcal{D}^{F_K(\cdot)}(1^\lambda) = 1] - Pr[\mathcal{D}^{R(\cdot)}(1^\lambda) = 1]| = \epsilon(\lambda),$$

where $K \xleftarrow{\$} (\mathbb{Z}_p)^{\ell+1}$, and R is a randomly chosen function from the set of functions with domain $\{0,1\}^\ell$ and image $\mathbb{G}$.

Theorem 1 (Theorem 4.1 of [35]). *If the DDH-Assumption holds, then F_K from Construction 1 has pseudo-random output.*

Observe that F_K from Construction 1 is *not* a pseudo-random function. The standard PRF textbook definition (which we omit here) requires indistinguishability of PRF output from output of a random function which F_K does not provide. However, F_K can trivially be converted into a PRF. If H_λ is a family of pairwise independent hash functions, and $h \xleftarrow{\$} H_\lambda$, then $\hat{F}_K(\cdot) = h(F_K(\cdot))$ is a PRF by a standard argument of the leftover hash lemma [19]. We will use the same argument later for our techniques and thus concentrate only on the pseudo-randomness property of Definition 1.

Definition 2 (OPRF). *Let F_K be a pseudo-random function family. An* OPRF *is a 2-party protocol between a sender and a receiver realizing the following ideal functionality. A trusted third party receives a key $K \in \{0,1\}^\lambda$ from the sender and input $x \in \{0,1\}^\ell$ from the receiver and sends $F_K(x)$ to the receiver.*

Construction 2 ($\text{OPRF}_K(x)$ from [16])**.** During initialization, sender S chooses key $K = (\alpha_0, \ldots, \alpha_\ell)$ by randomly sampling $\ell + 1$ scalars $\alpha_i \xleftarrow{\$} \mathbb{Z}_p$. To evaluate receiver R's input $x = (x_1 \ldots x_\ell)$, parties perform the following steps.

1. S randomly selects $(r_1, \ldots, r_\ell), r_i \xleftarrow{\$} \mathbb{Z}_p$.
2. S and R engage in ℓ rounds of $\binom{2}{1}$-OT. In round i, the server's input to OT is $(r_i, r_i \cdot \alpha_i)$, and the receiver's input is x_i. So, depending on x_i, the receiver gets either $z_i = r_i$ or $z_i = r_i \cdot \alpha_i$.
3. S sends $\hat{g} = g^{\frac{1}{\prod_{i=1}^{\ell} r_i}}$ to R.
4. R outputs $\text{OPRF}_K(x) = \hat{g}^{\prod_{i=1}^{\ell} z_i}$.

Freedman et al. [16] present a sketch for a proof of Construction 2. Effectively, this OPRF assembles the Naor and Reingold function F_K on input x in ℓ rounds. So, if the DDH assumptions holds, and if the underlying OT is secure and does not simultaneously leak r_i and $r_i \cdot \alpha_i$ at the same time, then Construction 2 is a secure OPRF (in the semi-honest model).

3 iPRF and iOPRF Definition

In this paper we introduce the notion of iterative pseudo-random functions (iPRF) and iterated oblivious pseudo-random functions (iOPRF).

Informally, an iPRF is a keyed function with bit strings $x = (x_1 \dots x_\ell)$ of length ℓ as input. It outputs ℓ bit strings v_i, each of length λ. Besides that each v_i is indistinguishable from a randomly chosen bit string, the crucial property which we target is that, for two bit strings x and x' sharing the same length k bit prefix, the first k outputs $(v_1, \dots, v_k)$ of iPRF will be the same.

Similar to OPRFs, an iOPRF is a two party protocol, where a receiver gets $\mathsf{iPRF}_K(x)$ for their input x, and the sender with input key K does not learn x. However, unlike standard OPRFs, iOPRFs run in ℓ rounds as required by the application scenarios we consider. In round i, the receiver adaptively inputs x_i such that eventually they receive all ℓ outputs from $\mathsf{iPRF}_K(x)$, where $x = (x_1 \dots x_\ell)$ is as specified during the ℓ rounds.

3.1 iPRF

Definition 3 (iPRF). *For inputs $x = (x_1 \dots x_\ell) \in \{0,1\}^\ell$ and randomly chosen keys $K = (K_1, \dots, K_\ell) \in \{0,1\}^{\ell \cdot \lambda}$, an* iterative pseudo-random function *family* $\mathsf{iPRF}_K(x)$ *is a sequence of function families*

$$\mathsf{iPRF}_K(x) = (f^1_{K_1}(x_1), \dots, f^\ell_{K_1,\dots,K_\ell}(x_1 \dots x_\ell)),$$

where each $f^i_{K_1,\dots,K_i}(x_1 \dots x_i) : \{0,1\}^{i\cdot\lambda} \times \{0,1\}^i \to \{0,1\}^\lambda$ is a pseudo-random function family with key $(K_1, \dots, K_i)$ from K and input $(x_1 \dots x_i)$ from x. Concatenated output $V_\lambda = v_1 || \dots || v_\ell, v_i = f^i_{K_1,\dots,K_i}(x_1 \dots x_i)$ is a family of random variables (a probability ensemble) of bit strings of length $\ell \cdot \lambda$.

Definition 3 implies that each probability ensemble $v_i = \{(v_i)_\lambda\}_{\lambda\in\mathbb{N}}$ of length λ bit strings is computationally indistinguishable from an ensemble u_i describing uniformly random bit strings of length λ. However, probability ensemble $V_\lambda = v_1 || \dots || v_\ell$ is *not* indistinguishable from an ensemble of uniformly random bit strings of length $\lambda \cdot \ell$. Instead, if any two inputs x and x' share the same prefix of length i, then the first i outputs $(v_1, \dots, v_i)$ of $\mathsf{iPRF}_K(x)$ will equal those of $\mathsf{iPRF}(x')$.

Besides being PRFs, we do not require anything else from underlying functions f^i. Note that, in general, PRFs do not need to be length-preserving [17].

// Let iPRF be an iterative pseudo-random function family

for $i = 1$ **to** ℓ **do**

  $R \to \mathsf{TTP} : x_i$;

  $S \to \mathsf{TTP}$: K_i; // $K = (K_1, \ldots, K_\ell)$

  $\mathsf{TTP} \to R : v_i$ such that $(v_1, \ldots, v_\ell) = \mathsf{iPRF}_K(x_1 \ldots x_\ell)$;

end

Fig. 1. Ideal Functionality $\mathcal{F}_{\mathsf{iOPRF}}$

Strawman Constructions Observe that the hashed Naor and Reingold PRF $\hat{F}_K$ from Construction 1 is not an iPRF and cannot easily be converted into an iPRF. First, to support $\lambda \cdot \ell$ outputs, λ for each input bit x_i, one might try and create an iPRF out of $(\hat{F}_{K_1}(x_1), \ldots, \hat{F}_{K_1,\ldots,K_\ell}(x_1 \ldots x_\ell))$, where $K_1 = \alpha_1, \ldots, K_\ell = \alpha_\ell$. However, this is in fact not an iPRF, as exemplified by inputs like $x = (10 \ldots 0)$. There, we have $\hat{F}_{K_1}(1) = \hat{F}_{K_1,K_2}(10) = \ldots = \hat{F}_{K_1,\ldots,K_\ell}(10 \ldots 0)$, so the output repeats starting from the 2^{nd} invocation of $\hat{F}_K$. In general, for any input $x = \mathsf{PREFIX}||0 \ldots 0$ ending with a sequence of zeros, $\hat{F}_K(x)$ will be equal to $\hat{F}_K(\mathsf{PREFIX})$ in violation of Definition 3.

A simple construction from existing symmetric key PRFs for an iPRF could be based on variable input length PRFs such as HMAC and a collision resistant hash function H. For example, consider $\mathsf{iPRF}_K(x) = (\mathsf{HMAC}_{H(K_1)}(x_1), \ldots, \mathsf{HMAC}_{H(K_1||\ldots||K_\ell)}(x_1 \ldots x_\ell))$. While this HMAC-based setup, and probably also adoptions of others PRFs like PRG-based PRFs [18], might result in valid iPRFs, we dismiss them in favor of our new construction Construction 3 (Section §4.1). Construction 3 offers several advantages: first, it builds on the Naor and Reingold pseudo-randomness which allows an elegant, formal security reduction from DDH to the iPRF property of Construction 3. More importantly, its key advantage is that you can use it as a building block to construct an efficient iOPRF which moreover supports, delegation, verifiability, and malicious security as we will see below.

3.2 iOPRF

Definition 4 **(π_{iOPRF}).** *Let* iPRF_K *be an iterative pseudo-random function family. An* iterative oblivious pseudo-random function *is an ℓ-round probabilistic protocol π_{iOPRF} between a sender S with input key $K \in \{0,1\}^{\lambda \cdot \ell}$ and receiver R with input bit string $x = (x_1 \ldots x_\ell) \in \{0,1\}^\ell$ with the following properties.*

- *Protocol π_{iOPRF} realizes the ideal functionality $\mathcal{F}_{\mathsf{iOPRF}}$ shown in Figure 1. This is a reactive functionality allowing queries from R in a total of ℓ rounds. After ℓ rounds, R has received $(v_1, \ldots, v_\ell) = \mathsf{iPRF}_K(x), |v_i| = \lambda$, from a trusted third party* TTP. *Sender S sends K_i in round i, but receives nothing from $\mathcal{F}_{\mathsf{iOPRF}}$.*

- *For all adversaries $\mathcal{A}$ in the real world, there exists a simulator* Sim_R *in the ideal world such that* R*'s view* $\mathsf{REAL}_{\pi_{\mathsf{iOPRF}},\mathcal{A},R}(x,K)$ *in the real world is computationally indistinguishable from* R*'s view* $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{iOPRF}},\mathsf{Sim}_R(x)}(x,K)$ *in the ideal world.*
- *For all adversaries $\mathcal{A}$ in the real world, there exists a simulator* Sim_S *in the ideal world such that* S*'s view* $\mathsf{REAL}_{\pi_{\mathsf{iOPRF}},\mathcal{A},S}(K)$ *in the real world is computationally indistinguishable from* S*'s view* $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{iOPRF}},\mathsf{Sim}_S}(K)$ *in the ideal world.*

The crucial difference of iOPRFs in contrast to regular OPRFs [4, 10, 16, 23, 25, 31] is that at the end of the protocol execution, R has received not one but ℓ PRF values v_i with $(v_1,\ldots,v_\ell) = \mathsf{iPRF}(x)$. For two inputs x and x' with the same length i bit prefix, values $v_1,\ldots,v_i$ will be the same. Note that receiver R can specify their input adaptively during ℓ rounds. Before sending x_i, R has learned v_{i-1} from $\mathcal{F}_{\mathsf{iOPRF}}$. Still, R receives output strings matching an iPRF, so they cannot combine outputs from different iOPRF executions with different input. For example, knowledge of $\mathsf{iOPRF}_K(10\ldots)$ and $\mathsf{iOPRF}_K(01\ldots)$ should not allow R to learn anything about $\mathsf{iOPRF}_K(11\ldots)$. Against a fully-malicious receiver, this cannot be accomplished easily with regular OPRFs. One might try and run ℓ instances of the OPRF, but the challenge is that one would have to force R to link their input during the i^{th} instance of the OPRF to the $(i-1)^{\text{th}}$ instance. Our iOPRF in Section §4.3 offers a solution to this challenge.

Verifiability An important aspect of OPRFs which we also require for iOPRFs is that of *verifiablity*, see Jarecki et al. [25] for technical details. Essentially, verifiablity implies that S proves to R that R's output $(v_1,\ldots,v_\ell)$ has been computed correctly. Towards providing malicious security, verifiability is especially important when the iOPRF is run multiple times, as S could cheat by using different keys for different protocol runs.

We refer to [25] for a treatment with more formal definitions in the context of OPRFs which also hold for iOPRFs. For our constructions, we will prove that R's output has been correctly computed by using a key which S has been committed to before.

Observe that the original Freedman et al. [16] OPRF (Construction 2) is not maliciously secure and thus does not offer verifiablity. Even if OT as a building block would be secure against a malicious adversary, it is unclear how to verify that the sender has used the same key K for different OPRF protocol runs.

Efficiency The last crucial property we require is that iOPRFs are efficient with respect to their communication and computational complexity. Efficiency is important in practice, as a client can perform $q \geq 1$ queries to decrypt q paths in the owner's data structure. For each query, after all ℓ rounds, an iOPRF has output ℓ bit strings of length security parameter, so the data exchanged between S and R and the number of computations involved to realize the iOPRF should be linear in ℓ.

The communication and computational complexities of an iOPRF are asymptotically *optimal* if, after any q queries, they are both in $O(q \cdot \ell)$. Our main contribution (Construction 5, §4.3) has optimal communication and computational complexities.

3.3 Delegation for iPRFs and iOPRFs

Informally, a PRF F with domain D is delegatable, if for some subset $D' \subset D$ you can (efficiently) compute a sub-key K' from key K and another PRF F' from F, such that $F'_{K'}$ equals F_K on all $x \in D'$, but is random everywhere else. There exists a rich theory on delegatable PRFs, see Kiayias et al. [28] for details.

In the context of iPRFs, we are particularly interested in delegating iterative PRF computation for strings $x = (x_1 \dots x_\ell)$ sharing the same fixed prefix. That is, a party P_1 knowing key K specifies a prefix $x^* = (x_1^* \dots x_i^*)$, computes K' and iPRF', and gives (iPRF', K') to party P_2. Party P_2 is then capable of computing $\mathsf{iPRF}_K(x)$ for all bit strings x having the same prefix x^*. At the same time, for all bit strings x with a different prefix than x^*, K' does not help P_2 in distinguishing the first i outputs of $\mathsf{iPRF}_K(x)$ from the output of random bit strings. We formalize this intuition in Definition 5.

Definition 5. *Let* iPRF *be an iterative pseudo-random function on length* ℓ *bit input strings with random key* K*. We call an* iPRF delegatable*, iff*

1. *There exists a PPT transformation algorithm* T*, which on input* $(\mathsf{iPRF}, K, x_1^* \dots x_i^*)$ *outputs* (iPRF', K')*, where* $\mathsf{iPRF}' : \{0,1\}^{\lambda\cdot(\ell-i)} \times \{0,1\}^{\ell-i} \rightarrow \{0,1\}^{\lambda\cdot(\ell-i)}$ *and* $\forall x' = (x'_1 \dots x'_{\ell-i}) : \mathsf{iPRF}'_{K'}(x') = \mathsf{iPRF}_K(x_1^* \dots x_i^* x'_1 \dots x'_{\ell-i})$.
2. *For all PPT distinguishers* $\mathcal{D}$ *and randomly chosen* K*, there exists a negligible function* ϵ *such that for sufficiently large* λ *we have*

$$\forall x^* = (x_1^* \dots x_i^*), \forall x = (x_1 \dots x_\ell), x_1 \dots x_i \neq x_1^* \dots x_i^* :$$
$$|Pr[(v_1, \dots, v_\ell) = \mathsf{iPRF}_K(x) : \mathcal{D}(1^\lambda, \mathsf{iPRF}', K', x, v_1, \dots, v_i) = 1] -$$
$$Pr[(r_1, \dots, r_i) \xleftarrow{\$} U_\lambda : \mathcal{D}(1^\lambda, \mathsf{iPRF}', K', x, r_1, \dots, r_i) = 1]| = \epsilon(\lambda),$$

where U_λ *is the probability ensemble of random bit strings of length* λ*,* K *is a randomly chosen key for* iPRF*, and* (iPRF', K') *are output by* $T(\mathsf{iPRF}, K, x_1^* \dots x_i^*)$.

Along the same lines, a *delegatable* iOPRF is an iOPRF where the underlying iPRF supports delegation.

Discussion Note that knowledge of K' and the first i values of the output $(v_i, \dots, v_i)$ of $\mathsf{iPRF}_K(x)$ does permit P_2 to enumerate all suffixes of strings x which share the same length i prefix as x. At first, this property might look like a severe restriction to the value of this type of delegation, but we will show in Section 6 that it has very interesting real-world applications.

We implicitly require non-triviality (bandwidth efficiency [28]) of delegation. For example, P_1 could delegate the capability to evaluate all strings with prefix x^* by simply computing $\mathsf{iPRF}_K(x)$ for all strings x with prefix x^* and sending the output to P_2. So, tuple (iPRF', K') should be smaller in size than the concatenation of all strings with prefix x^*.

Finally, we point out that delegation can be extended from iPRFs to iOPRFs in the natural way. If P_1 gives (iPRF', K') to P_2, then P_2 is also able to run a 2-party protocol with another party P_3, where P_3 correctly receives $\mathsf{iOPRF}'_{K'}(x') = \mathsf{iPRF}'_{K'}(x')$ for input x' with prefix x^* while P_2 learns nothing about x'.

4 New Constructions

We present our new constructions for both iPRF and iOPRF. To ease readability, we omit an important technicality in the description and proofs: our iPRF and iOPRF constructions do not output sequences of pseudo-random bit strings of length λ, but pseudo-random elements of DDH group $\mathbb{G}$. Yet, converting elements to bit strings follows from a standard application of the leftover hash lemma [19]. As $|p| \geq \lambda$, we have $|\mathbb{G}| \geq 2^\lambda$, and we silently assume in the following that each party implicitly hashes the output of iPRF and iOPRF using any pairwise independent family of hash functions.

4.1 iPRF Construction

Construction 3 (Our iPRF)**.** For any $\ell \in \mathbb{N}$, choose a random generator g and a key $K = (K_1, \ldots, K_\ell)$ by sampling ℓ pairs of random scalars $K_i = (\alpha_i, \beta_i) \xleftarrow{\$} (\mathbb{Z}_p)^2$. For any ℓ bit input $x = x_1 \ldots x_\ell$, we define function family $\mathsf{iPRF}_K(x_1, \ldots, x_\ell) = (f^1_{(\alpha_1,\beta_1)}(x_1), \ldots, f^\ell_{(\alpha_1,\beta_1),\ldots,(\alpha_\ell,\beta_\ell)}(x_1, \ldots, x_\ell))$, where

$$f^i_{(\alpha_1,\beta_1),\ldots,(\alpha_i,\beta_i)}(x_1 \ldots x_\ell) \stackrel{\text{def}}{=} g^{\prod_{x_i=1} \alpha_i \prod_{x_i=0} \beta_i} = g^{\prod_{j=1}^{i} \alpha_j^{x_j} \cdot \beta_j^{1-x_j}}.$$

Observe that you can also rewrite expression $g^{\prod_{j=1}^{i} \alpha_j^{x_j} \cdot \beta_j^{1-x_j}}$ as $g^{\prod_{j=1}^{i}(\alpha_j x_j + \beta_j(1-x_j))}$. This representation of f^i will be very useful during the presentation of our new techniques later.

iPRF Analysis To show that Construction 3 is actually an iPRF according to Definition 3, it is sufficient to show that each f^i is still a pseudo-random function.

Theorem 2. *If the DDH-Assumption holds, then for every $i \leq \ell$ and for every PPT distinguisher $\mathcal{D}$, there exists a negligible function ϵ such that for sufficiently large λ*

$$|Pr[\mathcal{D}^{f^i_{(\alpha_1,\beta_1),\ldots,(\alpha_i,\beta_i)}(\cdot)} = 1] - Pr[\mathcal{D}^{R^i(\cdot)} = 1]| = \epsilon(\lambda),$$

where the $(\alpha_1, \ldots, \beta_1), \ldots, (\alpha_i, \beta_i)$ are chosen randomly as in Construction 3, and R^i is a randomly chosen function from the set of functions with domain $\{0,1\}^i$ and image $\mathbb{G}$.

Proof. This follows because f^i is essentially taking the output from the PRF in Construction 1 and adding additionally adding extra random exponents, which maintains its character as a PRF. We can show this via reduction.

First, fix any $i \leq \ell$ and consider f^i. We prove the claim by reduction, showing that if $\mathcal{D}$ exists which can distinguish between f^i and a random function R^i, then we can build $\mathcal{D}'$ which can distinguish between F_K from Construction 1 (on i bit inputs and i element keys) and a random function R (on i bit inputs). This would violate F_K's pseudo-random output property of Definition 1.

Assume that $\mathcal{D}$ exists that can violate the inequality from Theorem 2. We create $\mathcal{D}'$ as follows. First, $\mathcal{D}'$ creates and stores a uniformly random sequence $(\beta_1, \ldots, \beta_\ell)$ as in Construction 3. Additionally, it queries its oracle for $g' = PRF(0)$ which is g^{α_0} if it is interacting with the real instance. This will be given to $\mathcal{D}$ as the generator so that $\mathcal{D}'$ can use results from its oracle, which will always include α_0, to satisfy queries from $\mathcal{D}$.

$\mathcal{D}'$ then runs $\mathcal{D}$ as a subroutine. Each time $\mathcal{D}$ queries the oracle for an evaluation on input $y \in \{0,1\}^i$, $\mathcal{D}'$ does the following:

1. Query their own oracle on input y and receive back z.
2. Calculate $z' = z^{\prod_{y_i=0} \beta_i}$.
3. Return z' to $\mathcal{D}$.

Eventually, $\mathcal{D}'$ outputs the same as $\mathcal{D}$. If $\mathcal{D}'$ is interacting with PRF F_K, then the z' values $\mathcal{D}'$ gives to $\mathcal{D}$ will be identical to function f^i, due to $\mathcal{D}'$ being able to multiply in the extra β components. If $\mathcal{D}'$ is interacting with a real random function, then the responses they give to $\mathcal{D}$ will be distributed identically to a random function, since z is the result of a random function and $\mathcal{D}'$ is raising it to fixed powers. Therefore, if $\mathcal{D}$ has a distinguishing advantage, so will $\mathcal{D}'$. $\mathcal{D}'$ has the same advantage that $\mathcal{D}$ does, rendering the reduction tight.

Delegation We achieve delegation for Construction 3 using the following transformation algorithm T.

$$\begin{aligned} \text{On input: } & (g, ((\alpha_1, \beta_1), \ldots, (\alpha_\ell, \beta_\ell)), x_1^* \ldots x_i^*), \\ T \text{ outputs: } & (g', ((\alpha_{i+1}, \beta_{i+1}), \ldots, (\alpha_\ell, \beta_\ell))), \\ & \text{where } g' = g^{\prod_{j=1}^{i} \alpha_j^{x_j} \cdot \beta_j^{1-x_j}}. \end{aligned}$$

Observe that g' is effectively a precomputed partial-iPRF for input $(x_1^* \ldots x_i^*)$. So, if party P_1 sends $(g', ((\alpha_{i+1}, \beta_{i+1}), \ldots, (\alpha_\ell, \beta_\ell)))$ to P_2, P_2 can then compute iPRF outputs $(v_{i+1}, \ldots, v_\ell)$ for any input string $x = (x_1 \ldots x_\ell)$ which has $(x_1^* \cdots x_i^*)$ as a prefix by computing $v_k = g'^{\prod_{j=i+1}^{k} \alpha_j^{x_j} \cdot \beta_j^{1-x_j}}$.

Lemma 1. *Construction 3 with transformation T is a delegatable* iPRF.

Proof. We prove this by straightforward reduction. Let iPRF_K be Construction 3 for inputs x of length $\ell + 1$ bits, and let $\widehat{\mathsf{iPRF}}_{\widehat{K}}$ be Construction 3 for inputs

x of length ℓ bits. Let prefix x^* be any length ℓ bit string, and K and $\widehat{K}$ are randomly chosen keys.

Assume there exists distinguisher $\mathcal{D}$ which can distinguish the first ℓ outputs from iPRF_K with a prefix different from x^* with non-negligible probability from ℓ random bit strings.

We build distinguisher $\mathcal{D}'$ who will be able to distinguish the ℓ outputs from $\widehat{\mathsf{iPRF}}_{\widehat{K}}$ from ℓ randomly chosen bit strings.

1. If $\mathcal{D}$ queries for delegation of length ℓ prefix x^*, $\mathcal{D}'$ will query their challenger for x^* and will get back z which is either $(v_1, \ldots, v_\ell) = \mathsf{iPRF}_K(x^*)$ or ℓ random bit strings $(r_1, \ldots, r_\ell)$.
2. $\mathcal{D}'$ generates a random pair $(\alpha_{\ell+1}, \beta_{\ell+1}) \xleftarrow{\$} (\mathbb{Z}_p)^2$. It computes transformation $(g' = z, (\alpha_{\ell+1}, \beta_{\ell+1}))$ and sends it to $\mathcal{D}$.
3. When $\mathcal{D}$ queries for x with a different prefix than x^*, $\mathcal{D}$ forwards x to their challenger, forwards the response to $\mathcal{D}$ and outputs whatever $\mathcal{D}$ outputs.

If $\mathcal{D}'$ is receiving the output of a $\widehat{\mathsf{iPRF}}_{\widehat{K}}$, then the values it gives to $\mathcal{D}$ will be identifically distributed to correct outputs of a delegated iPRF, with the effective key of K concatenated with the random $(\alpha_{\ell+1}, \beta_{\ell+1})$. If $\mathcal{D}'$ is receiving random strings $(r_1, \ldots, r_\ell)$, then $\mathcal{D}$ is also getting random strings. Therefore, $\mathcal{D}$'s view is distributed identically to its distinguishing game. If $\mathcal{D}$ has a non-negligible advantage in distinguishing, then $\mathcal{D}'$ will have the same advantage in distinguishing iPRF output from random strings.

4.2 Warm-Up: iOPRF with One-Sided Security

Our iPRF from Construction 3 can be computed as an iOPRF with only one-sided security, i.e., malicious receiver or semi-honest (or malicious, but only focusing on violating privacy [20]) sender, using a similar approach as the OPRF by Freedman et al. (Construction 2). Let $\mathsf{OT}(b, y_0, y_1)$ denote any $\binom{2}{1}$ oblivious transfer protocol which is one-sided simulatable [20] or even maliciously secure [2, 6]. Sender S holds y_0 and y_1 from $\mathbb{Z}_p$, receiver R holds $b \in \{0, 1\}$, and R obliviously retrieves y_b from S. Let $x = (x_1, \ldots, x_\ell)$ be R's input. Our first OT-based construction for a π_{iOPRF} protocol gives an iOPRF with one-sided security and works as follows.

Construction 4 (One-Sided Secure iOPRF)**.**

- S generates ℓ random scalars $r_i \xleftarrow{\$} \mathbb{Z}_p$.
- For each $1 \le i \le \ell$, R and S execute $\mathsf{OT}(x_i, r_i\beta_i, r_i\alpha_i)$, and R stores the result as z_i.
- S sends to R the sequence $C = (C_1, \ldots, C_\ell)$ where $C_i = g^{\frac{1}{\prod_{j=1}^{i} r_j}}$.
- R recovers iPRF output sequence $(v_1, \ldots, v_\ell)$ by calculating $v_i = C_i^{\prod_{j=1}^{i} z_j}$.

Correctness: For all $1 \le i \le \ell$, we have

$$\begin{aligned} v_i = C_i^{\prod_{j=1}^{i} z_i} &= g^{\frac{1}{\prod_{j=1}^{i} r_j} \cdot \prod_{j=1}^{i} z_j} = g^{\frac{1}{\prod_{j=1}^{i} r_j} \cdot \prod_{j=1}^{i} (\alpha_j r_j)^{x_j} (\beta_j r_j)^{1-x_j}} \\ &= g^{\prod_{j=1}^{i} \alpha_j^{x_i} \beta_j^{1-x_i}}. \end{aligned} \tag{1}$$

To prove security for Construction 4, we could make a similar argument as Freedman et al. [16], but rely on a one-sided simulatable OT. However, we refrain from presenting more details, as this iOPRF anyways provides only one-sided security and conversion to malicious security would be difficult. One would need to prove correct computation of the C_i and expensive maliciously secure OT with ZK proofs that the sender's input $(r_i\beta_i, r_i\alpha_i)$ matches previous commitments to α_i and β_i. This is very different from standard committed or verifiable OT [13, 24, 29].

4.3 Construction 5: DH-based iOPRF

We now present a new π_{iOPRF} protocol which realizes the ideal iOPRF functionality $\mathcal{F}_{\mathsf{iOPRF}}$ from Figure 1.

High-Level Intuition In round i of the ℓ rounds, sender S will receive two ciphertexts V_i and D_i from receiver R. During the course of the protocol, one of these ciphertexts will contain the iOPRF output and one acts as a "dummy", to keep S from learning input bits x_i of R. They are interchanged between rounds depending on the input bits.

For each round, using the i^{th} round's keys (α_i, β_i), S will then "apply" α_i to V_i and β_i to D_i, and send the results back to R. In preparation for the next round $(i+1)$, if $x_{i+1} \neq x_i$, R will swap V_i and D_i for the next round. After ℓ rounds, V_ℓ will have the keys applied which correspond to the input bits of R, and D_ℓ will have the complementary combination of keys applied. V_0 is initialized as an encryption of 1, so V_ℓ will contain the correct iOPRF output, whereas D_0 is initialized as an encryption of 0 so it will not contain any information.

We now turn to technical details.

Preliminaries Let there be two generators g_1, g_2 of prime order p group $\mathbb{G}$ where the DDH assumption holds. Neither party should know the discrete log of one generator g_i to the basis of the other generator $g_{j \neq i}$, which is true with high probability if they are chosen at random.

Elgamal Encryption We will use additive Elgamal encryption with private keys $sk \in \mathbb{Z}_p$ and public keys $pk = g_1^{sk}$. Ciphertext c to encrypt $m \in \mathbb{Z}_p$ is $c = (c[0], c[1]) = (g_1^r, pk^r \cdot g_2^m) \leftarrow \mathsf{Enc}_{pk}(m)$, where $r \xleftarrow{\$} \mathbb{Z}_p$.

Pedersen Commitments A Pedersen commitment $\mathsf{com}(m) \in \mathbb{G}$ to message $m \in \mathbb{Z}_p$ is defined as $\mathsf{com}(m) = g_1^r \cdot g_2^m$, where $r \xleftarrow{\$} \mathbb{Z}_p$. To open $\mathsf{com}(m)$, reveal tuple (m, r). Pedersen commitments are perfectly hiding and computationally binding.

For some input string $x = (x_1 \dots x_\ell)$, we define the output of π_{iOPRF} for the receiver as $(v_1, \dots, v_\ell)$ with $v_i = g_2^{\prod_{j=1}^{i}(\alpha_j x_j + \beta_j(1-x_j))}$.

We describe details of Construction 5 by its formal π_{iOPRF} interface (Definition 4), i.e., first its initialization and then its iterative processing.

π_{iOPRF} Initialization Sender S randomly chooses secret key $K = ((\alpha_1, \beta_1), \dots, (\alpha_\ell, \beta_\ell)), (\alpha_i, \beta_i) \xleftarrow{\$} (\mathbb{Z}_p)^2$. Receiver R computes a random Elgamal private key $sk \xleftarrow{\$} \mathbb{Z}_p$ and public key $pk = g_1^{sk}$, and sends pk to S. Receiver R proves knowledge of sk using a standard Schnorr ZK proof of knowledge (see §4.4).

Receiver R computes $V_0 \leftarrow \mathsf{Enc}_{pk}(1)$ and $D_0 \leftarrow \mathsf{Enc}_{pk}(0)$, sends them to S and proves that these are encryptions of 1 and 0 (see §4.4 below).

π_{iOPRF} Iterative Processing in ℓ Rounds In round $i \in \{1, \dots, \ell\}$, for S' input bit x_i:

1. **Receiver shuffles:**
   (a) For input bit x_i, R computes Pedersen commitment $\mathsf{com}(x_i)$ and proves that $x_i \in \{0,1\}$ (see §4.4). Similarly, R computes $\mathsf{com}(1-x_i)$ and proves that $(1-x_i) \in \{0,1\}$ (see §4.4). Finally, R proves that the sum of plaintexts behind $\mathsf{com}(x_i)$ and $\mathsf{com}(1-x_i)$ equals 1 (see §4.4).
   (b) Receiver R chooses $r, r', r'', r''' \xleftarrow{\$} \mathbb{Z}_p$ and computes Elgamal ciphertexts
   $$\begin{aligned} c_i &= (g_1^r \cdot V_{i-1}[0]^{x_i}, pk^r \cdot V_{i-1}[1]^{x_i}) \\ c'_i &= (g_1^{r'} \cdot V_{i-1}[0]^{1-x_i}, pk^{r'} \cdot V_{i-1}[1]^{1-x_i}) \\ d_i &= (g_1^{r''} \cdot D_{i-1}[0]^{x_i}, pk^{r''} \cdot D_{i-1}[1]^{x_i}) \\ d'_i &= (g_1^{r'''} \cdot D_{i-1}[0]^{1-x_i}, pk^{r'''} \cdot D_{i-1}[1]^{1-x_i}) \end{aligned}$$
   and sends (c_i, c'_i, d_i, d'_i) to S.
   (c) Receiver R proves correctness of the above computations in ZK. Specifically, (c_i, c'_i, d_i, d'_i) result from correct exponentiation with x_i (or $1-x_i$) from $\mathsf{com}(x_i)$ (or $\mathsf{com}(1-x_i)$), and multiplication with a random power of g_1 and pk, i.e., re-randomization (homomorphic addition of encryption of 0). See §4.4 below for details.
   Both parties compute
   $$T_i = (c_i[0] \cdot d'_i[0], c_i[1] \cdot d'_i[1]) \qquad U_i = (c'_i[0] \cdot d_i[0], c'_i[1] \cdot d_i[1]).$$

   In the first round, after this step, T_1 is an encryption of 1 and U_1 is an encryption of 0 iff $x_1 = 1$. Iff $x_1 = 0$, then T_1 is an encryption of 0 and U_1 is an encryption of 1. However, sender S does not know which of the two is the case.
2. **Sender commits:** Sender S computes Pedersen commitments $(\mathsf{com}(\alpha_i), \mathsf{com}(\beta_\ell))$, sends them to R, and proves knowledge of plaintexts in ZK (see §4.4).

3. **Sender computes PRF:** For $r, r' \xleftarrow{\$} \mathbb{Z}_p$, S computes the two Elgamal ciphertexts

$$X_i = (g_1^r \cdot T_i[0]^{\alpha_i}, pk^r \cdot T_i[1]^{\alpha_i}) \qquad Y_i = (g_1^{r'} \cdot U_i[0]^{\beta_i}, pk^{r'} \cdot U_i[1]^{\beta_i}),$$

   sends (X_i, Y_i) to R, and proves correct exponentiation (scalar multiplication of plaintexts) with α_i and β_i coming from previous commitments $\mathsf{com}(\alpha_i)$, $\mathsf{com}(\beta_i)$ *and* re-randomization of ciphertexts (see §4.4).
4. **Receiver shuffles back:** For $r, r', r'', r''' \xleftarrow{\$} \mathbb{Z}_p$, R computes

$$\begin{array}{ll} P_i = (g_1^r \cdot X_i[0]^{x_i}, pk^r \cdot X_i[1]^{x_i}) & P'_i = (g_1^{r'} \cdot X_i[0]^{1-x_i}, pk^{r'} \cdot X_i[1]^{1-x_i}) \\ Q_i = (g_1^{r''} \cdot Y_i[0]^{x_i}, pk^{r''} \cdot Y_i[1]^{x_i}) & Q'_i = (g_1^{r'''} \cdot Y_i[0]^{1-x_i}, pk^{r'''} \cdot Y_i[1]^{1-x_i}) \end{array}$$

   and sends (P_i, P'_i, Q_i, Q'_i) together with ZK proofs of correct computation (see §4.4) to S.
   Both S and R compute $V_i = (P_i[0] \cdot Q'_i[0], P_i[1] \cdot Q'_i[1])$ and $D_i = (P'_i[0] \cdot Q_i[0], P'_i[1] \cdot Q_i[1])$.
   In round i, after this step, V_i is an encryption of $\mathsf{iPRF}_K(x_1, \ldots, x_i)$, and U_i is an encryption of 0. When computing T_{i+1} and U_{i+1}, these values will be used instead of the encryptions of 0 and 1 and the iterative computation of the PRF continues. Since both parties compute V_i and U_i, R cannot cheat and substitute for a value of his choice.
5. Receiver R computes and outputs one iPRF value $v_i = \frac{V_i[1]}{V_i[0]^{sk}}$.

Discussion Observe that, in the last step, R can never decrypt additively homomorphic Elgamal ciphertext $(V_i[0], V_i[1])$ and thus compute an α_i or β_i. As α_i or β_i are in the exponent and due to the hardness DLOG, R can only compute $v_i = g_2^{\ldots\alpha_i\ldots}$ or $v_i = g_2^{\ldots\beta_i\ldots}$.

If R wants to run several execution of Construction 5 and wants that S uses the same key, then R will verify that commitments sent by S in Step (2) do not change between executions. This leads to verifiability.

Also note that communication complexity and computational complexity are both in $O(\ell)$ per query, i.e., they are asymptotically optimal.

4.4 Security Analysis

We prove security of Construction 5 using simulation in the standard model. The simulation uses several efficient Zero-Knowledge Proofs of Knowledge hybrids introduced first. To ease readability, we actually present Honest-Verifier Zero-Knowledge (HVZK) versions of the proofs, but one can convert these to maliciously verifier Zero-Knowledge proofs of knowledge using the following two general transformations [21]. We stress that we have evaluated and benchmarked the full malicious verifier ZK proofs of knowledge in Section 5, i.e., including the two transformations.

Zero Knowledge (instead of Honest-Verifier ZK) All our efficient ZK proofs below are three-move ("Sigma") ZK proofs. Recall that a three-move ZK proof comprises messages (t, e, s), where first message t is a commitment from P sent to V, e is V's challenge sent to P, and s is the final message sent from P to V.

To make these proofs zero-knowledge instead of only HVZK, we send an additional message before first message t of the regular three-move proof. In this new first message, V sends a Pedersen commitment $\mathsf{com}(e) = g_1^r \cdot g_2^e$ to their random challenge e to V. The proof then continues with V sending their regular commitment t of the regular three-move proof and V opening $\mathsf{com}(e)$ by sending (e, r). If $\mathsf{com}(e)$ matches (e, r), P finally sends last message s of the regular proof. Verifier V accepts, if t and s of the regular proof match e.

This technique allows a simulator Sim simulating P to cheat in the ZK proof. More specifically, after receiving $\mathsf{com}(e)$, Sim internally computes a valid ZK proof (t', e', s'), assuming a random challenge e'. Sim sends t' to V and receives (e, r). If (e, r) matches $\mathsf{com}(e)$, Sim rewinds V to the point after V has sent $\mathsf{com}(e)$. Knowing e, Sim computes a t and s, such that (t, e, s) will be accepted by V. How exactly t and s are chosen depends on the statement we want to prove, but are typically straightforward for the Schnorr-style proofs we use below. We show an example in §4.4.

Witness Extraction for Pedersen Commitments To transform our ZK proofs to ZK proofs of knowledge, we rely on the extractability of commitments. Pedersen commitments are trapdoor commitments which means that a party knowing a trapdoor ρ can open a commitment $\mathsf{com}(\cdot)$ to any plaintext they want (equivocable). We use this property for witness extraction in three-move ZK proofs as follows.

Before starting the actual ZK proof by the first message t from the prover to the verifier, we send the following two messages.

1. Prover P sends to verifier V: $\hat{g} = g_1^\rho$ for random $\rho \xleftarrow{\$} \mathbb{Z}_p$.
2. Verifier V will use this $\hat{g}$ instead of g_2 for the computation of the commitment to challenge e. That is, V computes and sends back commitment $\mathsf{com}(e) = g_1^r \cdot \hat{g}^{\,e}$ for their random challenge $e \in \mathbb{Z}_p$ as in the previous section.

The ZK proof then continues as usual with P sending t and V opening $\mathsf{com}(e)$ by sending (e, r). If (e, r) match $\mathsf{com}(e)$, P sends final message s and ρ to V. Only if both is correct, the last ZK proof message s matches P's commitment t and challenge e, and ρ matches $\hat{g} = g_1^\rho$, V accepts.

This setup enables a simulator Sim simulating V to extract the witness from P. After receiving trapdoor ρ from P, Sim rewinds P until after the point were P sends t to V. Knowing trapdoor ρ, Sim can open $\mathsf{com}(e)$ to any $e' \neq e$ they want by solving $r + \rho \cdot e = r' + \rho \cdot e'$ for r', i.e., they compute $r' = r + \rho \cdot (e - e')$. Running two executions of the ZK proof with the same input and messages from P but different challenges extracts the witness of the ZK proof. Details on which

e to send in each execution again depend on the exact three-move ZK proof, but are typically obvious. We refer to Hazay and Lindell [21] for more details.

In conclusion, these two transformation will render our three-move ZK proofs below into (fully-maliciously secure) ZK proofs of knowledge. We name each proof below with a hybrid which we will use in the main proof later. So, for example, the hybrid for the proof of encryption is called $f_{\mathsf{zk}}^{\mathsf{enc}}$.

$f_{\mathsf{zk}}^{\mathsf{enc}}$: Proof of Encryption/Commitment to m To prove that an encryption $c = (c[0], c[1]) = (g_1^r, pk^r) \leftarrow \mathsf{Enc}_{pk}(0)$ is an encryption of $m = 0$, P proves that $(g_1, c[0], pk, c[1])$ is a DDH tuple. You can prove that tuple $(u_1 = g_1, u_2 = g_1^r, u_3 = g_1^{sk}, u_4 = g_1^{sk \cdot r})$ is a DDH tuple using the Chaum and Pedersen [12] protocol as follows.

1. P sends $(t_1 = u_1^\rho, t_2 = u_3^\rho)$ for $\rho \xleftarrow{\$} \mathbb{Z}_p$ to V.
2. V sends $e \xleftarrow{\$} \mathbb{Z}_p$ to P.
3. P sends $s = \rho + e \cdot r$ to V.
4. V accepts if $u_1^s = u_2^e \cdot t_1$ and $u_3^s = u_4^e \cdot t_2$.

This proof has an important property. Instead of showing that some ciphertext encrypts $m = 0$, we can easily generalize it to show encryption of arbitrary m. Specifically, we set $c'[1] = \frac{c[1]}{g_2^m}$ and run the proof with $m = 0$ for new Elgamal ciphertext $(c[0], c'[1])$.

Finally, observe that Pedersen commitments are similarly structured as the right-hand side $c[1]$ of an Elgamal ciphertext, just without the secret key. Thus, to prove a Pedersen commitment $\mathsf{com}(m)$ to m, parties divide $\mathsf{com}(m)$ by g_2^m and run a Schnorr proof for r used in the commitment (P sends $t = g_1^\rho$, V sends e, P sends $s = \rho + e \cdot r$, and V accepts if $g_1^s \stackrel{?}{=} \frac{\mathsf{com}(m)^e}{g_2^m} \cdot t$.)

$f_{\mathsf{zk}}^{\mathsf{pop}}$: Proof for Knowledge of Plaintext For $\mathsf{com}(m) = g_1^r \cdot g_2^m$, prover P can prove that they know m.

1. P sends $t = g_1^{\rho_1} \cdot g_2^{\rho_2}$ for $\rho_1, \rho_2 \xleftarrow{\$} \mathbb{Z}_p$ to V.
2. V sends $e \xleftarrow{\$} \mathbb{Z}_p$ to P.
3. P sends $s_1 = \rho_1 + e \cdot r$ and $s_2 = \rho_2 + e \cdot m$ to V.
4. V checks whether $g_1^{s_1} \cdot g_2^{s_2} \stackrel{?}{=} \mathsf{com}(m)^e \cdot t$.

$f_{\mathsf{zk}}^{\mathsf{bit}}$: Proof of Plaintext Bit For a commitment $\mathsf{com}(x_i)$, prover P can prove that x_i is a bit, i.e., $x_i \in \{0, 1\}$. This is an application of the *one-out-of-two* (OR) technique [22]. Essentially, P proves that either $x_i = 1$ which implies proving that $\mathsf{com}(x_i)$ equals $g_1^{r_1} \cdot g_2$ for some r_1, or $x_i = 0$ which implies proving that $\mathsf{com}(x_i)$ equals $g_1^{r_2}$ for some r_2. Proving that $\mathsf{com}(x_i)$ equals $g_1^{r_1} \cdot g_2$ is equivalent to proving that $\frac{\mathsf{com}(x_i)}{g_2}$ equals $g_1^{r_1}$.

P will prove that they know (I) an r such that $g_1^r = \frac{\mathsf{com}(x_i)}{g_2}$ or (II) an r such that $g_1^r = \mathsf{com}(x_i)$. These are essentially two standard Schnorr proofs. The trick is that P chooses e_1 and e_2 such that, for the verifier's challenge e, we have $e = e_1 + e_2$. Prover P proves knowledge of r_1 for (I) using challenge e_1 and knowledge of r_2 for (II) using challenge e_2. Thus, P can choose either e_1 or e_2 before sending their first message of the ZK proof and cheat in one proof. Without loss of generality, let $x_i = 1$, so P will cheat in proof (II). This works as follows.

1. P sends $t_1 = g_1^{\rho_1}$ and $t_2 = \mathsf{com}(x_i)^{-e_2} \cdot g_1^{s_2}$, where $\rho, s_2 \overset{\$}{\leftarrow} \mathbb{Z}_p$, to V.
2. V sends $e \overset{\$}{\leftarrow} \mathbb{Z}_p$ to P.
3. P calculates $e_1 = e - e_2$, sends $e_1, e_2, s_1 = \rho_1 + e_1 \cdot r$, and s_2 to V.
4. V checks $e \overset{?}{=} e_1 + e_2$, $g_1^{s_1} \overset{?}{=} \left(\frac{\mathsf{com}(x_i)}{g_2}\right)^{e_1} \cdot t_1$ and $g_1^{s_2} \overset{?}{=} \mathsf{com}(x_i)^{e_2} \cdot t_2$.

$\boldsymbol{f_{\mathsf{zk}}^{\mathsf{sum}}}$**: Proof of Sum of Plaintexts equals 1** For commitments $\mathsf{com}(x) = g_1^r \cdot g_2^x$ and $\mathsf{com}(1-x) = g_1^{r'} \cdot g_2^{1-x}$, P shows that the sum of plaintexts equals 1.

1. P and V compute $\mathsf{com}(1) = \mathsf{com}(x) \cdot \mathsf{com}(1-x) = g_1^{r+r'} \cdot g_2$.
2. P proves that $\mathsf{com}(1)$ is a commitment to 1 (see §4.4).

$\boldsymbol{f_{\mathsf{zk}}^{\mathsf{ExR}}}$**: Proof of Exponentiation and Re-Encryption** One can efficiently prove correctness of combinations of linear operations in one step. We present the example for the correctness of exponentiation of two elements (A, B) from group $\mathbb{G}$ with a committed value x and then multiplying A^x by $g_1^{r'}$ and B^x by $pk^{r'}$ from our protocol. So, this can be used to prove correct exponentiation (homomorphic scalar multiplication) of an Elgamal ciphertext by a previously committed scalar x and subsequent re-randomization of the result (homomorphic addition of Elgamal encryption of 0).

Specifically, given two group elements (A, B) and commitment $\mathsf{com}(x) = g_1^r \cdot g_2^x$, prove correctness that $(C = g_1^{r'} \cdot A^x, D = pk^{r'} \cdot B^x)$ are the result of exponentiation with x and multiplying with $g_1^{r'}$ and $pk^{r'}$, $r' \overset{\$}{\leftarrow} \mathbb{Z}_p$, known to P.

1. P sends $t_1 = g_1^{\rho_1} \cdot A^{\rho_2}, t_2 = pk^{\rho_1} \cdot B^{\rho_2}, t_3 = g_1^{\rho_3} \cdot g_2^{\rho_2}$ to V.
2. V sends $e \overset{\$}{\leftarrow} \mathbb{Z}_p$ to P.
3. P sends $s_1 = \rho_1 + e \cdot r'$, $s_2 = \rho_2 + e \cdot x$, and $s_3 = \rho_3 + e \cdot r$ to V.
4. V checks whether $g_1^{s_1} \cdot A^{s_2} \overset{?}{=} C^e \cdot t_1$, $pk^{s_1} \cdot B^{s_2} \overset{?}{=} D^e \cdot t_2$, and $g_1^{s_3} \cdot g_2^{s_2} \overset{?}{=} \mathsf{com}(x)^e \cdot t_3$.

Proof of Construction 5 We now turn to our main proof, showing that Construction 5 is a secure iOPRF. We prove in the hybrid model, using ZK hybrids with their abbrevations as introduced in the previous section. Recall that, in the hybrid model, ZK hybrids are run by separate trusted third parties. Yet, during simulation, it is the simulator who takes the role of the TTP and thus automatically gets the adversary's inputs and can also cheat, see Lindell [33] for details.

Theorem 3. *Assume that Construction 3 is an iterative pseudo-random function family* $\mathsf{iPRF}_K(\cdot)$*. Then, Construction 5 is an* iOPRF*, realizing functionality* $\mathcal{F}_{\mathsf{iOPRF}}$ *in the* $\left(f_{\mathsf{zk}}^{\mathsf{enc}}, f_{\mathsf{zk}}^{\mathsf{pop}}, f_{\mathsf{zk}}^{\mathsf{bit}}, f_{\mathsf{zk}}^{\mathsf{sum}}, f_{\mathsf{zk}}^{\mathsf{ExR}}\right)$ *hybrid-model.*

Proof. First, observe that Construction 5 is correct. Let x be the receiver's input, and K the key chosen by the sender. If both sender and receiver are honest, then the sender outputs nothing, and the receiver outputs $(v_1, \ldots, v_\ell) = \mathsf{iPRF}_K(x)$. Thus, we focus on proving security and build simulators for two cases: one where S is compromised, and one where R is compromised.

We will show that a simulator Sim can be constructed from both the perspective of S and R such that the adversary $\mathcal{A}$'s view is indistinguishable from real executions of the protocol. Thus we show that neither a compromised S nor a compromised R learn anything from the real execution of Construction 5 beyond what is specified by the ideal functionality in Figure 1.

In our presentation below, we will use the term "Sim *aborts"* as a shorthand for Sim sending abort to the TTP, simulating its party aborting to $\mathcal{A}$, and then outputting whatever $\mathcal{A}$ outputs.

In both cases below, the simulator will faithfully act as a verifier for ZKPs when interacting with $\mathcal{A}$ as necessary, aborting if the proof does not verify correctly. We omit these messages for readability since they require no special knowledge or behavior from the simulator. Our strategy will broadly be to:

- Replace Elgamal ciphertexts sent by R with encryptions of zero (arbitrarily chosen). Due to Elgamal's IND-CPA property, these ciphertexts will be indistinguishable from the real protocol for $\mathcal{A}$. Since S reveives no output from the real execution of the protocol, ciphertexts do not have to conform to any expectations.
- Replace computation of X_i and Y_i by S in the real protocol with an encryption of the output of the iOPRF received from the TTP. Sim does not know $K_i = (\alpha_i, \beta_i)$ and so cannot faithfully compute X_i or Y_i, but it knows from the TTP what output v_i should. Consequently, Sim crafts these values accordingly to simulate the real protocol and "cheat" in ZKPs where Sim acts as the prover (see, e.g., § 4.4).

Together, this will allow the simulator to generate a view which is indistinguishable from a real execution, thus proving that our construction is secure according to Definition 4.

Note that also for all ZKPs with Sim as a prover, Sim acts as the TTP and "cheats" to convince $\mathcal{A}$. In many instances, Sim could honestly prove to $\mathcal{A}$, so "cheating" is not really. Yet, for ease of exposition, we assume that all proofs are simulated this way.

Case 1: We assume that $\mathcal{A}$ has compromised S and build simulator Sim taking the role of S in the ideal world, internally simulating a receiver to $\mathcal{A}$ which it only has black box access to.

Sim starts $\mathcal{A}$ and generates an Elgamal key pair (sk, pk), sends pk to $\mathcal{A}$, and simulates $f_{\mathsf{zk}}^{\mathsf{enc}}$. Also, Sim generates $V_0 = \mathsf{Enc}_{pk}(0)$ and $D_0 = \mathsf{Enc}_{pk}(0)$, sends them to $\mathcal{A}$, and simulates $f_{\mathsf{zk}}^{\mathsf{enc}}$.

During the i^{th} round,

1. Sim sends two independent commitments of zero and simulates $f_{\mathsf{zk}}^{\mathsf{bit}}$ and $f_{\mathsf{zk}}^{\mathsf{sum}}$.
2. Sim also computes and sends (c_i, c'_i, d_i, d'_i), all encryptions of zero, to $\mathcal{A}$ and simulates $f_{\mathsf{zk}}^{\mathsf{ExR}}$.
3. Sim receives $(\mathsf{com}(\alpha_i), \mathsf{com}(\beta_i)$ from $\mathcal{A}$. Sim also receives (α_i, β_i) together with random coins from $f_{\mathsf{zk}}^{\mathsf{pop}}$ sent from $\mathcal{A}$ to $f_{\mathsf{zk}}^{\mathsf{pop}}$. If these do not match the commitments, Sim *aborts*.
4. Sim receives (X_i, Y_i) from $\mathcal{A}$ as well as (α'_i, β'_i) and random coins from $f_{\mathsf{zk}}^{\mathsf{ExR}}$. If $\alpha_i \neq \alpha'_i$ or $\beta_i \neq \beta'_i$ or if random coins do not match computations specified in Construction 5, then Sim *aborts*. If they match, Sim forwards $K_i = (\alpha_i, \beta_i)$ to the TTP.
5. Sim sends P_i, P'_i, Q_i, Q'_i, encryptions of zero, to $\mathcal{A}$ and simulates $f_{\mathsf{zk}}^{\mathsf{ExR}}$.

Sim outputs what $\mathcal{A}$ outputs.

During simulation, whenever $\mathcal{A}$ aborts, Sim also *aborts*.

Indistinguishable views In the protocol, there are three types of messages that Sim sends to $\mathcal{A}$: Pedersen commitments, Elgamal ciphertexts, and ZKP messages. All of the Elgamal ciphertexts are freshly encrypted (or re-encrypted) using fresh randomness. They are thus indistinguishable from any other Elgamal encryption, regardless of any a priori knowledge that $\mathcal{A}$ might have. As stated above, the ZKPs are simulated and are thus also indistinguishable from a real execution. Finally, the commitments are perfectly hiding and are never revealed during the protocol, so they are also indistinguishable from the commitments of a real execution.

Case 2: We assume that $\mathcal{A}$ has compromised R and build simulator Sim as follows.

Sim starts $\mathcal{A}$ and receives pk from $\mathcal{A}$ and (sk', pk') from $f_{\mathsf{zk}}^{\mathsf{enc}}$ which $\mathcal{A}$ has sent. If $pk \neq pk'$ or $g_1^{sk'} \neq pk$, Sim *aborts*. Also, Sim receives (V_0, D_0) from $\mathcal{A}$ and $\mathcal{A}$'s random coins from $f_{\mathsf{zk}}^{\mathsf{enc}}$. If random coins do not match encryptions of 1 (V_0) or 0 (D_0), Sim *aborts*.

During the i^{th} round,

1. Sim receives $(\mathsf{com}(x_i), \mathsf{com}(1-x_i))$ from $\mathcal{A}$ and $(x'_i, 1-y'_i)$ with the commitments' random coins from $f_{\mathsf{zk}}^{\mathsf{bit}}$. If x'_i or $1-y'_i$ and random coins do not match commitments, Sim *aborts*. In the same way, Sim receives z and a random coin for the commitment from sum hybrid $f_{\mathsf{zk}}^{\mathsf{sum}}$. If $z \neq 1$ or $z \neq x'_i + 1 - y'_i$ or the random coin does not match the commitment, Sim *aborts*. If everything matches, Sim knows $\mathcal{A}$'s input $(x_i, 1 - x_i)$.
   Sim receives (c_i, c'_i, d_i, d'_i) from $\mathcal{A}$ and random coins and $(x'_i, 1 - y'_i)$ from $f_{\mathsf{zk}}^{\mathsf{ExR}}$. If $(x'_i, 1 - y'_i)$ do not match the ones from the previous step or if any of the computations do not match (c_i, c'_i, d_i, d'_i), Sim *aborts*.
   Sim computes (T_i, U_i) as in Construction 5.
2. Sim selects random $(\alpha'_i, \beta'_i) \xleftarrow{\$} (\mathbb{Z}_p)^2$, commits to them, sends them to $\mathcal{A}$, and proves knowledge of (α'_i, β'_i) using $f_{\mathsf{zk}}^{\mathsf{pop}}$.

Table 1. Cost breakdown

ℓ	CPU (ms)		Communication (kB)		Total runtime (ms)	
	Sender	Receiver	Sender	Receiver	LAN1	WAN1
5	44	41	11.7	26.1	171	2425
10	88	81	23.2	51.4	325	4571
15	126	123	34.6	76.6	512	6707
20	174	162	46.1	101.9	679	8873
25	218	202	57.5	127.1	836	10987
30	267	248	69.0	152.4	968	13128

3. Sim queries the TTP for x_i' and gets back v_i. If $x_i = 1$, Sim sets $X_i \leftarrow \mathsf{Enc}_{pk}(v_i)$ and $Y_i \leftarrow \mathsf{Enc}_{pk}(0)$. If $x_i = 0$, Sim sets $X_i \leftarrow \mathsf{Enc}_{pk}(0)$ and $Y_i \leftarrow \mathsf{Enc}_{pk}(v_i)$. Sim sends (X_i, Y_i) to $\mathcal{A}$ and *cheats* in $f_{\mathsf{zk}}^{\mathsf{ExR}}$, convincing $\mathcal{A}$ that (X_i, Y_i) are the result of raising T_i and U_i to α_i' and β_i' and then re-encrypting.
4. Finally, Sim receives (P_i, P_i', Q_i, Q_i') from $\mathcal{A}$ and random coins and $(x_i', 1-y_i')$ from $f_{\mathsf{zk}}^{\mathsf{ExR}}$. Again, Sim verifies correct computation of (P_i, P_i', Q_i, Q_i') and whether $(x_i', 1-y_i')$ match previously received values. If anything does not match, Sim *aborts*.
   Sim computes (V_i, D_i) as in Construction 5.

Sim outputs what $\mathcal{A}$ outputs.

During simulation, whenever $\mathcal{A}$ aborts, also Sim *aborts*.

Indistinguishable views As before, the commitments are perfectly hiding and are not revealed and so are indistinguishable from commitments of a real protocol execution. ZKPs are also simulated as before and are indistinguishable for the same reason.

The only part that is different in this case is the returned values of X_i and Y_i, which have to decrypt to the correct output of the iOPRF in order to match the real protocol. Fortunately, Sim can query the TTP for the correct output and generate encryptions that match that output. In the real protocol, S reencrypts X_i and Y_i before returning them to R, and so they are indistinguishable from the fresh encryptions generated by Sim.

As R verifies whether S sends the same commitments to (α_i, β_i) during multiple executions of Construction 5, we trivially achieve verifiability.

5 Implementation

To show practicality of Construction 5 including its ZK proofs, we have implemented and benchmarked its performance in several realistic network settings. We stress that our implementation is a full implementation of Construction 5, with all Zero-Knowledge Proofs of Knowledge of Section 4.4, i.e., including witness extractability and security against fully malicious verifiers. Sender and receiver instances communicate via standard TCP sockets.

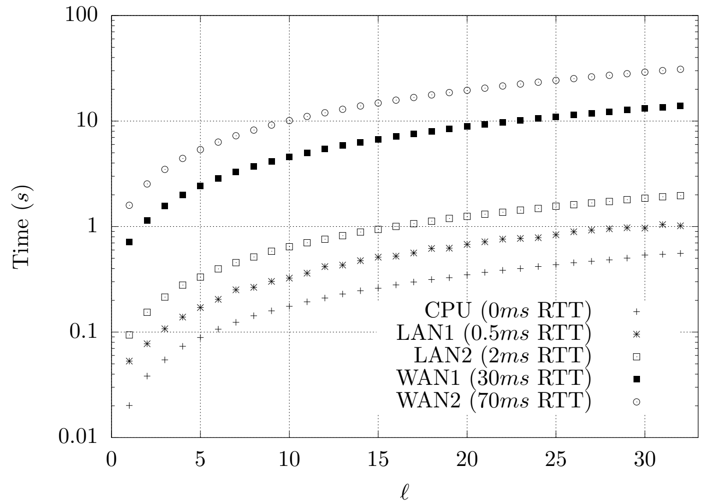

Fig. 2. Total runtime of Construction 5

Our implementation is done in C and uses OpenSSL for elliptic curve operations on NIST curve `secp224r1`. The source code is available for download [1]. We benchmark our implementation on a 4.1 GHz Core i5-10600k. As network latency is typically the bottleneck in multi-round secure two-party computation protocols, we benchmark Construction 5 in different settings with different network latencies. To precisely control network latency between sender and receiver instances, we use Linux' standard `tc-netem` tool. Figure 2 shows benchmark results averaged over 50 executions, and Table 1 presents the cost breakdown.

We measure total run time for values of ℓ ranging from 1 to 32. Note that $\ell = 32$ would support binary tree data structures with 2^{32} different paths and a total of $2^{33} - 1$ ($\approx$ 8.6 billion) nodes. We vary latency assuming LAN scenarios with standard Gigabit Ethernet (0.5 ms RTT) or WiFi (2 ms RTT) and WAN scenarios for intra-continental communication (30 ms RTT) and inter-continental communication (70 ms RTT) [36]. We also show an evaluation with 0 ms RTT, however even this number is still dominated by the TCP communication overhead. We found that the computation alone in our protocol, including all EC computation and ZKPs, is approximately 3 ms per iOPRF iteration.

Each iOPRF evaluation for a tree data structure with 2^{20} nodes needs about 170 ms of CPU time per party with our (unoptimized) implementation. As soon as we introduce higher latency, CPU time contributes very little to total runtime and communication latency becomes the main performance obstacle. For example, in the WAN1 scenario with intra-continental communication between sender and receiver, total runtime is about 9 s of which less than 4% is spent with actual computation, and the remainder is consumed by network latency.

We conclude from Figure 2 that even for large values of ℓ and for high latency network connections, Construction 5 has only a few seconds of runtime which is practical for many scenarios.

Discussion: Performance of related approaches iOPRFs must be interactive, requiring an interaction per iteration, and interactivity turns out to be the runtime bottleneck. Yet, we argue that such interaction is still more efficient than alternatives.

For example, we could construct a single round iOPRF protocol using fully homomorphic encryption (FHE). However, we would then have to evaluate ℓ one-way functions inside the FHE circuit and prove their correct computation. We expect such computations would be too long to be practical even on very powerful hardware. Another alternative would be general cryptographic primitives which allow iterative one-way functions. Recent Multi-Linear Maps could be used for this purpose. However, there exist no secure multi-linear map for generic constructions, let alone efficient ones. Lastly, the sender could compute the iPRF for all possible inputs by the receiver and the receiver could select one using oblivious transfer. Another example of obliviously evaluating such a function are distributed point functions [5] which would avoid oblivious transfer. However, in both cases the server would need to evaluate 2^ℓ functions rendering this approach quickly infeasible. In conclusion, our iOPRF avoids the pitfalls of non-interactive design alternatives providing practical performance.

Finally, one could envision realizing an iOPRF using general maliciously MPC frameworks such as MP-SPDZ [27] or efficient maliciously secure 2PC [37]. However, it is sender-receiver interactivity which turns out to be the main challenge. Evaluation of an arithmetic (SPDZ) or Boolean (2PC) circuit cannot be stopped, its output revealed, and then continued with new input. Instead, sender and receiver would need to securely evaluate ℓ different circuits. After evaluating circuit i, the receiver learns the i^{th} output, and specifies the $(i+1)^{\text{st}}$ input, and both parties evaluate another circuit. Inside the circuit, the sender and receiver would need to somehow prove to each other that they continue the evaluation with correct data which is not trivial. For example, the circuit would need to output an (encrypted) state to the sender after each iteration which the circuit then verifies in the next round based on additional information output to the receiver. The sender would also need to prove that they are using the same key as one they have committed to, previously. Recall that evaluation of cryptographic primitives inside a circuit is very expensive, even using fast maliciously secure 2PC. For example, Wang et al. [37] report 85 ms for the evaluation of a single SHA2 circuit (amortized over 1024 circuits) in a scenario with latency comparable to LAN1. This is already more expensive than one full round of Construction 5.

6 Applications

An immediate application of our iOPRF is to force correct compliance of clients in structured encryption by allowing them to only query a contiguous path in the graph data. This can be accomplished by adding a layer of encryption inside of existing structured encryption solutions such that each data element is also encrypted with a key derived from one iteration of the iOPRF. After the structured encryption protocol is complete, an iOPRF protocol is executed which will allow for final decryption of the results only if they are on a contiguous path.

To hide from the server which path is queried, the client can fetch each node using Private Information Retrieval or maliciously secure OT.

Also in scenarios with structured encryption, the iOPRF's delegation feature can be used to delegate control over well-specified sub-trees of the original data to other parties. The delegate can then act as a data owner on their sub-tree, serving requests from clients with the same security property as the original data owner.

To understand specifically the usefulness of iOPRFs, we now consider a specific implementation of RFID tag authentication which uses a limited form of structured encryption.

6.1 RFID

Radio Frequency IDentification (RFID) applications comprise a large quantity of RFID tags attached to precious goods and RFID readers which are connected to a central backend database. The goal is that readers can properly identify tags using wireless communication in the presence of adversaries.

An adversary observing wireless tag-reader interaction or being able to interact with tags themselves should not be able to identify or trace tags or even fabricate new tags or clone tags to counterfeit goods. The main technical challenge is that RFID tags are extremely resource restricted and can merely compute a cryptographic hash function. While readers can perform more powerful operations, they typically feature low storage (no state), but have network connectivity, e.g., to connect to a central database. RFID security has been a very active area of research, see Juels [26] for an overview.

In a typical scenario, the reader wants to know whether a tag and therewith the good it is attached to is valid, by interacting first with the tag and then with the database. Typically, the tag stores a unique key, and the reader performs a challenge-response type of authentication, using the database which knows all tags' keys. However, previous work has assumed that database and readers are within the same trust domain, as the database learns which tag the reader is querying for. This is an unnecessary strong and often unrealistic requirement. To protect tag privacy and internal details of supply or distribution chains, the database should not learn which tag the reader is querying for. For example, if several readers successively query for the same tag, the database knows that a specific tag has traveled between these readers. At the same time, the database does not want to give unrestricted access to the reader or allow queries which would leak more information than necessary for the identification of a single tag per query. If the reader would receive more information, the danger would be that a reader fabricates tags.

To mitigate these problems, we show how we can extend a prominent RFID security protocol from the literature, the one by Molnar et al. [34], by a simple application of our iOPRF.

High-Level Idea In the original work by Molnar et al., the database prepares a binary key tree of height ℓ storing random keys in nodes. Leaves in the tree are enumerated by their path from the root to the leaf. For example, the left most leaf is represented by the bit string of ℓ zero bits. A tag is uniquely identified by its ID, a bit string $x = (x_1 \dots x_\ell)$. During initialization, a tag with ID x receives all keys from the root to the leaf represented by x. During tag identification, the tag chooses a random r, "encrypts" r with each of their keys, and sends the resulting sequence of ciphertexts to the reader. The reader can access the database and query keys. The reader checks which path in the tree decrypts and ends up with a specific ID (leaf). As you can see, this protocol does not protect the tag from a prying database. A simple solution of just sending the whole key tree to the reader might overburden the reader's storage capabilities and also impose a security risk: having access to all keys, the reader could fabricate an arbitrary number of tags.

Our modification to the Molnar et al. protocol simply consists of exchanging the way keys in the tree are computed. In our case, the keys are outputs of the iOPRF which will allow the reader to query the database for exactly one contiguous path. As a result, we hide from the database which tag the reader is querying for, and the database knows that the reader only gets one path of secrets from the tree and will be able to identify exactly one tag with it.

Technical Details Let $N = 2^\ell$ be the total number of tags in the system. Each tag uniquely corresponds to a leaf of a height ℓ binary *key tree*. To identify a tag, a reader can communicate with the database which stores all keys of the key tree.

Preliminaries The database knows a secret key K and populates binary key tree T as follows. First, nodes in this key tree are indexed by bit strings following the intuitive notation that the left child ("0") of some node indexed by bit string $\gamma_1 \dots \gamma_i$ is index by $\gamma_1 \dots \gamma_i 0$, and the right child ("1") is indexed by $\gamma_1 \dots \gamma_i 1$. By convention, the root is indexed by empty bit string ϵ.

Database Initialization Root node ϵ stores random key $K_\epsilon \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$. The left child of the root stores key $K_0 = \mathsf{iOPRF}_K(0)$, and the right child stores key $K_1 = \mathsf{iOPRF}_K(1)$. For a node $\gamma_1 \dots \gamma_i$, the left child stores key $K_{\gamma_1 \dots \gamma_i 0} = \mathsf{iOPRF}_K(\gamma_1 \dots \gamma_i 0)$, and its right child stores key $K_{\gamma_1 \dots \gamma_i 1} = \mathsf{iOPRF}_K(\gamma_1 \dots \gamma_i 1)$.

During authentication of tag x, the database will run $\mathsf{iOPRF}_K(\cdot)$ as the sender, and the reader will be the receiver with input bit strings $x = (x_1 \dots x_\ell)$ as follows.

Tag Initialization During initialization of a new tag x, the database stores a sequence of $(\ell+1)$ keys K on the tag: one for each node on the path from the root K_ϵ of tree T to leaf $K_x = K_{x_1\ldots x_\ell}$. The tag also stores its own ID x.

Secure Tag Identification Each tag identifies itself to a reader using a variation of the Molnar et al. protocol:

- For security parameter λ, tag x chooses $r \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and sends r together with a hash of r and each of their $(\ell+1)$ keys and the next bit, respectively. More formally, the tag sends

$$\begin{aligned}\mathsf{Trace} = (r, &T_0 = H(r, K_\epsilon, x_1), \ldots, T_\ell = H(r, K_{x_1\ldots x_{\ell-1}}, x_\ell),\\ &H(r, K_{x_1\ldots x_\ell})).\end{aligned}$$

  The difference to the original protocol is that we also include next bit x_i into each hash. This allows the reader to check which node to query for during the next iteration. Otherwise, the reader would have to retrieve both children of the current node, revealing "one more key" per level of the tree to the reader.
- The reader uses the iOPRF as the receiver and the database as the sender to identify the tag as follows.
  - The database begins by sending K_ϵ to the reader.
  - The reader checks whether either $H(r, K_\epsilon, 0)$ or $H(r, K_\epsilon, 1)$ matches T_0.
  - Depending on the outcome, the reader iteratively continues and queries either the left child ($H(r, K_\epsilon, 0)$ matches) or the right child ($H(r, K_\epsilon, 1)$ matches) of the root with the iOPRF, compute keys, checks which matches etc.

As you can see, the security we are aiming for asks only for a (delegatable) OPRF. Our iOPRF supports delegation, but can do more. We could also ask as an additional security requirement that the reader should only learn "one path", i.e., one tag per interaction with the database.

Security Analysis To summarize security requirements, we briefly describe a reactive, ideal functionality $\mathcal{F}$. The database sends their input, keys $K_\epsilon, K_0, K_1, \ldots, K_{1\ldots1}$, all $(2N-1)$ keys of the key tree, to a TTP, and the reader sends an empty bit string. Then, the TTP sends K_ϵ to the reader, and nothing to the database. The internal state s of TTP is initialized to the empty bit string. Then, the RFID reader and TTP additionally interact in a total of ℓ rounds. In round i, let the internal state be bit string $s = \gamma_1 \ldots \gamma_{i-1}$. The reader sends bit γ_i, and TTP responds with $K_{\gamma_1\ldots\gamma_i}$ and updates its state to $s = \gamma_1 \ldots \gamma_i$.

Lemma 2. *In the random oracle model, the modified Molnar et al. protocol securely realizes ideal functionality $\mathcal{F}$.*

As the proof of Lemma 2 is straightforward, we only summarize it in a draft.

Proof (Proof (Sketch)). We build a simulator for the case of a compromised reader. The simulator for the case of a compromised database works accordingly.

1. Simulator Sim begins by preparing an initially empty key-value table RO to implement a standard random oracle functionality $H(\cdot)$. During simulation, whenever any party calls $H(k)$ for some input k, this functionality will check whether pair (k, v) is already in table RO and responds with v in that case. Otherwise, H generates a random string v of length λ, sends v back to the caller, and places (k, v) in RO.
2. Also, Sim generates a random key $K = ((\alpha_1, \beta_1), \ldots, (\alpha_\ell, \beta_\ell))$ for iOPRF. Sim sends ϵ to TTP and receives K_ϵ which it forwards to $\mathcal{A}$.
3. Sim and $\mathcal{A}$ run Construction 5 with Sim as the sender and $\mathcal{A}$ as the receiver. During the i^{th} iteration of Construction 5:
   (a) Sim extracts $\mathcal{A}$'s input x_i from the Pedersen commitment, forwards it to TTP, and receives back $K_{x_1 \ldots x_i}$.
   (b) Sim adds key-value pair $(g_2^{\prod_{j=1}^{i} \alpha_j^{x_j} \beta_j^{1-x_j}}, K_{x_1 \ldots x_i})$ to table RO.

Observe that $\mathcal{A}$'s view in the simulation is indistinguishable from their view in a real protocol execution.

Note that $\mathcal{A}$ can perform an input-substitution attack, i.e., query for some path which does not match the tag they are currently interacting with. Without the ability to perform public key cryptography on the tag, the strongest security for the database one can guarantee is that the reader can get one path, identifying one tag and thus can fabricate or clone at most one tag per interaction.

Delegation As iOPRFs are delegatable, we also support scenarios where a main database delegates the information to identify tags of, e.g., different countries or regions to different sub-databases.

We abstain from presenting lengthy details, but such delegation with iOPRFs would bring the advantage that if keys from one regional sub-database are stolen and thus tags in that region can be fabricated, tags and their identification in other sub-databases are still secure.

6.2 Private Decision Tree Evaluation

Another application where we can apply an iOPRF is in the area of private evaluation of decision trees. There, the goal is to allow a client holding a feature vector to query an outsourced decision tree held by a server, resulting in the client receiving the machine learning classification of their feature vector without the owner of the decision tree learning what their input was. We refer to Kiss et al. [30] for an overview.

For example, the protocol by Wu et al. [38] accomplishes this with two main techniques:

1. Each node of the decision tree stores one value which will be compared against one feature of the client's feature vector. To enable this, the client encrypts their feature vector with additively homomorphic encryption using the client's public key and sends ciphertexts to the server. For each node of the tree, the server computes homomorphic DGK [14] comparisons "$<$" of one of the client's encrypted features with the specific node's value and sends encrypted comparison outcomes back to the client. Therewith, the client can identify the path in the tree and the leaf node corresponding to their input.
2. Once the correct leaf node is identified, the client uses oblivious transfer to retrieve it and compute the final classification.

This protocol works for a semi-honest client, however it does not prevent a malicious client from retrieving leaf nodes which do not actually correspond to the result of their classification. This is because the server is not able to verify that the client traverses a contiguous path in the tree or that the OT they perform corresponds to that path if they did.

Consequently, Wu et al. suggest an augmented version of the protocol that can handle malicious clients using a new *conditional oblivious transfer*, but a maliciously-secure version could also be constructed simply by replacing OT with our iOPRF.

Each node in the tree could be encrypted using keys derived from the iOPRF evaluation of their index, meaning that the client would have to traverse a path in the tree all the way to the leaf in order to decrypt it. The only necessary modification for this approach to work is a small number of additional ZKPs to "bind" the results of the homomorphic evaluation to the input of the iOPRF. When constructed this way, the client can use much more efficient (maliciously secure) private information retrieval [8] instead of the expensive conditional OT designed by Wu et al. [38]. For space reasons, we list only the main technical modifications necessary (in Appendix A).

7 Conclusion

In this paper, we have introduced the concept of an iteratable oblivious pseudorandom function and presented a construction which is provably secure in the standard model under the DDH assumption. We have fully implemented and evaluated this construction and shown that it is efficient in practice, comparable to similar protocols. We have also presented several applications for iOPRF protocols that demonstrate their usefulness, particularly in applications where (two-sided) malicious security is necessary.

Bibliography

[1] Anonymous. Source code, 2021. `https://www.dropbox.com/s/ujvhxexysy34xok/code.tgz?dl=0`.

[2] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More Efficient Oblivious Transfer Extensions. *J. Cryptol.*, 30(3):805–858, 2017.

[3] D. Boneh, H.W. Montgomery, and A. Raghunathan. Algebraic pseudorandom functions with improved efficiency from the augmented cascade. In *CCS*, pages 131–140, 2010.

[4] D. Boneh, D. Kogan, and K. Woo. Oblivious Pseudorandom Functions from Isogenies. In *ASIACRYPT*, pages 520–550, 2020.

[5] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. In *CCS*, pages 1292–1303, 2016.

[6] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation. In *CCS*, pages 291–308, 2019.

[7] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science*, 260, 1997.

[8] Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In *Australasian Conference on Information Security and Privacy*, pages 50–61. Springer, 2004.

[9] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, pages 577–594. Springer, 2010.

[10] M. Chase and P. Miao. Private Set Intersection in the Internet Setting from Lightweight Oblivious PRF. In *CRYPTO 2020*, volume 12172, pages 34–63, 2020.

[11] D. Chaum. Blind Signatures for Untraceable Payments. In *CRYPTO*, pages 199–203, 1982.

[12] D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In *CRYPTO*, volume 740, pages 89–105, 1992.

[13] C. Crepeau, J. van de Graaf, and A. Tapp. Committed Oblivious Transfer and Private Multi-Party Computation. In *CRYPTO*, volume 963, pages 110–123, 1995.

[14] I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *ACISP*, pages 416–430, 2007.

[15] Y. Dodis and A. Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In *PKC*, volume 3386, pages 416–431, 2005.

[16] M.J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword Search and Oblivious Pseudorandom Functions. In *TCC*, volume 3378, pages 303–324, 2005.

[17] O. Goldreich. *The Foundations of Cryptography – Volume 1: Basic Techniques*, chapter 3.6.4, pages 158–159. Cambridge University Press, 2001.

[18] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[19] J. Håstad, R. Impagliazzo, L.A. Levin, and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[20] C. Hazay and Y. Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In *TCC*, volume 4948, pages 155–175, 2008.

[21] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions.* Springer, 2010. ISBN 978-3-642-14302-1.

[22] B.A. Huberman, M.K. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *Conference on Electronic Commerce*, pages 78–86, 1999.

[23] S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*, volume 5444, pages 577–594, 2009.

[24] S. Jarecki and V. Shmatikov. Efficient Two-Party Secure Computation on Committed Inputs. In *EUROCRYPT*, volume 4515, pages 97–114, 2007.

[25] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In *ASIACRYPT*, volume 8874, pages 233–253, 2014.

[26] A. Juels. RFID security and privacy: a research survey. *IEEE J. Sel. Areas Commun.*, 24(2):381–394, 2006.

[27] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. In *CCS*, pages 1575–1590, 2020.

[28] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *CCS*, pages 669–684, 2013.

[29] M.S. Kiraz, B. Schoenmakers, and J. Villegas. Efficient Committed Oblivious Transfer of Bit Strings. In *Information Security, 10th International Conference, ISC*, volume 4779, pages 130–144, 2007.

[30] Á. Kiss, M. Naderpour, J. Liu, N. Asokan, and T. Schneider. SoK: Modular and Efficient Private Decision Tree Evaluation. *Proc. Priv. Enhancing Technol.*, 2019(2):187–208, 2019.

[31] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In *CCS*, pages 818–829, 2016.

[32] A.B. Lewko and B. Waters. Efficient Pseudorandom Functions from the Decisional Linear Assumption and Weaker Variants. In *CCS*, page 112–120, New York, NY, USA, 2009. ISBN 9781605588940.

[33] Y. Lindell. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[34] D. Molnar, A. Soppera, and D. A. Wagner. A Scalable, Delegatable Pseudonym Protocol Enabling Ownership Transfer of RFID Tags. In *SAC*, volume 3897, pages 276–290, 2005.

[35] M. Naor and O. Reingold. Number-theoretic Constructions of Efficient Pseudo-random Functions. In *FOCS*, pages 458–467, 1997.

[36] Verizon. IP Latency Statistics , 2021. `https://enterprise.verizon.com/terms/latency/`.

[37] X. Wang, S. Ranellucci, and J. Katz. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *CCS*, pages 21–37, 2017.

[38] D.J. Wu, T. Feng, M. Naehrig, and K. Lauter. Privately evaluating decision trees and random forests. *Proceedings on Privacy Enhancing Technologies*, 2016(4):335–355, 2016.

A Decision Trees

As an alternative to their paper, we summarize here the changes necessary to convert the semi-honest secure protocol from Wu et al. to a fully-malicious-secure version using iOPRF. We will reference our modifications in contrast with their protocol (Figure 1 in [38]).

1. In step 1, the client proves that their input encryptions are bits. This also happens in the maliciously-secure version from the original paper.
2. In step 2, the negation of the intended DGK comparisons [14] are also computed. This way the client has a "successful" comparison one way or the other to use in their proof to the server that they are behaving correctly.
3. In step 4, the server additionally encrypts each node of the tree with a symmetric key derived from an iOPRF. The keys are chosen such that each node can be decrypted by an iOPRF evaluation that corresponds to that node's location in the binary tree, adjusted for the randomly flipped comparisons. The goal here is to restrict the client to only being able to decrypt the nodes corresponding to the contiguous path in the tree resulting from its comparisons.
4. In step 5, the client uses PIR [8] to retrieve the target leaf node instead of conditional OT. The client additionally runs an iOPRF protocol to retrieve the symmetric key necessary to decrypt their chosen leaf node. In execution of this protocol, they also prove in ZK that the input to the iOPRF corresponds to the correct results of the comparison protocol (see Appendix A.1).

A.1 Binding Homomorphic Comparisons to iOPRF Input

Since the client now executes two DGK comparisons per level of the tree, the original intended one and its negation, they now always have a "successful" comparison at every level, which tells them which direction to go in the tree. The main idea behind the proofs that will bind the client to the correct path is that they can use the encryption of zero that results from a successful comparison as evidence to the server that they are going in the direction they are supposed to.

At each level of the tree $k \in [d]$, the client creates a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(0)$ and generates a commitment com to $x_k = 1$ if the comparison at that node was true and $x_k = 0$ if its negation was true. This corresponds to the direction their comparison at level k in the shuffled tree tells them to go on the next level. They then must prove that there exists an i such that c (the encryption of zero) is plaintext-equivalent to either $\mathsf{ct}_{k,i}$ or $\mathsf{ct}'_{k,i}$ (the result of the negated comparison), and that if it is $\mathsf{ct}_{k,i}$ then com is 1, or if it is $\mathsf{ct}'_{k,i}$ then com is 0. Then, $comm$ is used as the commitment in the iOPRF protocol. This binds the output of the comparison to the input of the iOPRF, completing the proof.

Let $a \equiv b$ signify that a and b are encryptions of the same value and $a \equiv 0, 1$ signify that a is an encryption of 0 or 1. The statement being proven can then be writen as follows

$$\left[(c \equiv \mathsf{ct}_{k,1} \vee \ldots \vee c \equiv \mathsf{ct}_{k,t}) \wedge \mathsf{com} \equiv 1\right]$$
$$\vee$$
$$\left[(c \equiv \mathsf{ct}'_{k,1} \vee \ldots \vee c \equiv \mathsf{ct}'_{k,t}) \wedge \mathsf{com} \equiv 0\right]$$

We do not produce a full ZK proof for this statement, as it can be efficiently designed in the same way we design ZK proofs in Section 4.4 (plaintext equivalence is equivalent to a proof of DDH tuple, one-out-of two technique for the $\vee$, parallel proofs for $\wedge$ etc.). For more details on efficient composition of ZK proofs, see also Camenisch and Stadler [7].