Delegatable Pseudorandom Functions and Applications

Aggelos Kiayias
Dept. Informatics & Telecom.
National and Kapodistrian U. of Athens, Greece
aggelos@di.uoa.gr

Stavros Papadopoulos
Dept. of Computer Science
Hong Kong U. of Science and Technology, Hong Kong
stavrosp@cse.ust.hk

Nikos Triandopoulos
RSA Laboratories
Cambridge, MA, USA
nikolaos.triandopoulos@rsa.com

Thomas Zacharias
Dept. Informatics & Telecom.
National and Kapodistrian U. of Athens, Greece
thzacharias@di.uoa.gr

Abstract

We put forth the problem of delegating the evaluation of a pseudorandom function (PRF) to an untrusted proxy and introduce a novel cryptographic primitive called *delegatable pseudorandom functions*, or DPRFs for short: A DPRF enables a proxy to evaluate a pseudorandom function (PRF) on a strict subset of its domain using a trapdoor derived from the DPRF secret key. The trapdoor is constructed with respect to a certain *policy predicate* that determines the subset of input values which the proxy is allowed to compute. The main challenge in constructing DPRFs is to achieve *bandwidth efficiency* (which mandates that the trapdoor is smaller than the precomputed sequence of the PRF values conforming to the predicate), while maintaining the pseudorandomness of unknown values against an attacker that adaptively controls the proxy. A DPRF may be optionally equipped with an additional property we call *policy privacy*, where any two delegation predicates remain indistinguishable in the view of a DPRF-querying proxy: achieving this raises new design challenges as policy privacy and bandwidth efficiency are seemingly conflicting goals.

For the important class of policy predicates described as (1-dimensional) *ranges*, we devise two DPRF constructions and rigorously prove their security. Built upon the well-known tree-based GGM PRF family [17], our constructions are generic and feature only logarithmic delegation size in the number of values conforming to the policy predicate. At only a constant-factor efficiency reduction, we show that our second construction is also policy private. Finally, we describe that their new security and efficiency properties render our DPRF schemes particularly useful in numerous security applications, including RFID, symmetric searchable encryption, and broadcast encryption.

1 Introduction

Due to its practical importance, the problem of securely delegating computational tasks to untrusted third parties comprises a particularly active research area. Generally speaking, secure delegation involves the design of protocols that allow the controlled authorization for an—otherwise untrusted—party to compute a given function while achieving some target security property (e.g., the verifiability of results or privacy of inputs/outputs) and also preserving the efficiency of the protocols so that the delegation itself remains meaningful. Beyond protocols for the delegation of general functionalities (e.g., [18, 33]) a variety of specific cryptographic primitives have been considered in this context (see related work in Section 2).

Quite surprisingly, pseudorandom functions (PRFs), a fundamental primitive for emulating perfect randomness via keyed functions which finds numerous applications in information security, have not been explicitly studied in the context of delegation of computations (of PRF values). Hereby, we initiate a study on this matter.

A new PRF concept. We introduce a novel cryptographic primitive, called *delegatable pseudorandom function* (DPRF), which enables the delegation of the evaluation of a PRF to an untrusted proxy according to a given predicate that defines the inputs on which the proxy will evaluate the PRF.

Specifically, let $\mathcal{F}$ be a PRF family, and $\mathcal{P}$ a set of predicates, called the *delegation policy*, defined over the domain of $\mathcal{F}$. A DPRF is a triplet $(\mathcal{F}, T, C)$ constructed with respect to $\mathcal{P}$, which provides the elementary functionality shown in Figure 1: For any secret key k and a predicate $P \in \mathcal{P}$, the delegator computes a trapdoor τ via algorithm T, and τ is transmitted to the proxy. The latter then runs algorithm C on τ to derive exactly the set of PRF values $B_P = \{f_k(x)|P(x)\}$, where $f_k \in \mathcal{F}$, i.e., the PRF value on every input x satisfying predicate P, overall correctly enabling the evaluation of PRF f_k subject to predicate P *without* explicit knowledge of secret key k, or even the input values $A_P = \{x|P(x)\}$.

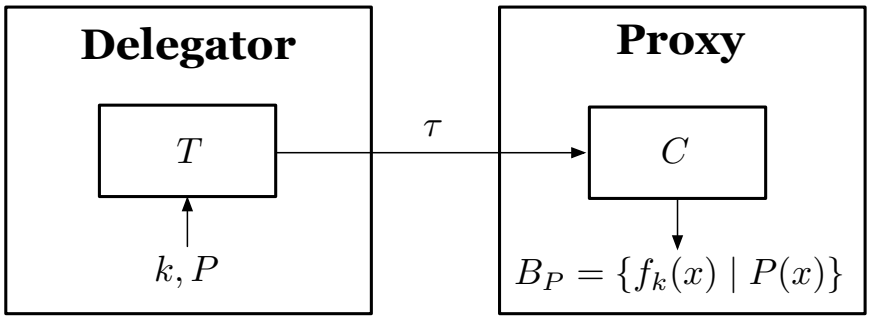


Figure 1: The DPRF functionality

What motivates the above scheme is *bandwidth efficiency*: As long as the trapdoor τ is sublinear in the size $|B_P|$ of delegated PRF values, the delegation is meaningful as the delegator conserves resources (otherwise, the proxy could be provided directly with all the PRF values B_P).

At the same time, the DPRF must retain the security properties of the underlying PRF, namely, (i) *pseudorandomness* for any value x conforming to the delegation predicate P, i.e., $P(x)$, and (ii) *unpredictability* for any nonconforming value x such that $\neg P(x)$. In addition, a DPRF can optionally satisfy a *policy privacy* property which prevents the proxy from inferring information about P or the delegated (input) set A_P from the trapdoor τ.

Our definitional framework. We introduce a formal definitional framework for the new primitive of

DPRFs, carefully capturing all the above technical requirements. We first rigorously define the *correctness* and *security* requirements that any DPRF should meet. Correctness captures the ability of the proxy to successfully evaluate the PRF on exactly those inputs specified by predicate P (i.e., set A_P). Security captures the requirement that the delegation capabilities of the PRF do not compromise its core pseudorandomness property. Note that this condition goes beyond the standard security definition of PRFs, since the pseudorandomness attacker may now adaptively query a trapdoor delegation oracle with predicates of its choice.

Equally important is also the *policy privacy* property that a DPRF may optionally satisfy, intuitively capturing the inability of a malicious proxy to learn any (non-trivial) property about the delegation set A_P (that is not implied by set B_P). Our security notion postulates that for any two predicates P, P' the corresponding trapdoors are indistinguishable, provided that $|A_P| = |A_{P'}|$ and no other PRF queries trivially separating V_P and $V_{P'}$ are posed by the adversary.

GGM-based realization for range predicates. We devise two *bandwidth-efficient* and *provably secure* DPRF constructions for the case where the delegation policy contains predicates described by *1-dimensional ranges*. Range-based policies is an interesting use case, as many applications maintain an ordering over the PRF inputs, and delegation rights are defined with respect to ranges of such inputs.

Our first DPRF scheme is called *best range cover*, or BRC for short, and relies on the well-known GGM PRF family [17]. This family defines a PRF based on the hierarchical application of any length-doubling pseudorandom generator (PRG) according to the structure induced by a tree, where input values are uniquely mapped to root-to-leaf paths. By exploiting the above characteristics, our BRC scheme features logarithmic delegation size in the number $|B_P|$ of values conforming to the policy predicate, by having the trapdoor τ comprise a subset I_P of *internal* PRG values that (almost optimally) cover the target range B_P of PRF values. We provide a formal security proof for the above scheme which, interestingly, does not trivially follow from the GGM security proof. The reason is that the adversary can now employ delegation queries to learn any internal PRG value in the tree, and not simply leaf PRF values as in the original security game in [17]. We note that, although similar "range-covering" GGM-based constructions appear in the literature, e.g., in [30], no formal security analysis has been given.

However, our BRC scheme does not satisfy policy privacy, as the structure of the PRG values in I_P leaks information about predicate P. This motivates our second construction, called *uniform range cover*, or URC for short. This scheme augments BRC in a way that renders all trapdoors corresponding to ranges of the same size indistinguishable. This is achieved by carefully having τ comprising a subset I'_P of PRG values that cover the target range B_P of PRF values less optimally: I'_P contains PRG values that are descendants of those values in I_P at a tree height that depends solely on $|B_P|$ (which by definition leaks to the adversary). More interestingly, by adopting the above change, URC retains both the asymptotic logarithmic bandwidth complexity of BRC and its DPRF security, but it crucially achieves a policy privacy notion appropriately relaxed to our setting of range predicates. Inherently, as we argue, no efficient tree-based DPRF scheme can satisfy policy privacy, so we have to resort to a sufficient relaxation which we call *union policy privacy*.

Main applications. Finally, our DPRF schemes, equipped with bandwidth efficiency, security and policy privacy (URC only), lend themselves to new schemes that provide scalable solutions to a wide range of information security and applied cryptography settings involving the controlled authorization of PRF-based computations. Generally, DPRFs are particularly useful in applications that rely on the evaluation of (secret) key-based cryptographic primitives on specific inputs (according to an underlying policy): Using a DPRF scheme then allows a cost-efficient, yet secure and private, key management for an untrusted proxy who is otherwise capable in executing a particular computational task.

We outline several such applications in which DPRFs are useful, including authentication and access

control in RFIDs, efficient batch querying in searchable encryption, as well as broadcast encryption. Due to the underlying GGM building component, our DPRFs are extremely lightweight, as their practical implementation entails a few repeated applications of *any* efficient candidate instantiation of a PRG that is length doubling.

Summary of contributions. Our main contributions are:

- We initiate the study of policy-based delegation of the task of evaluating a pseudorandom function on specific input values, and introduce the concept of *delegatable* PRFs (DPRFs).
- We develop a general and rigorous definitional framework for the new DPRF primitive, capturing properties such as *bandwidth efficiency*, *correctness*, *security* and *policy privacy*, and also offering a relaxed *union policy privacy* that is arguably necessary for tree-wise DPRF constructions.
- We present a framework for building DPRF schemes for the important case where the delegation policy is governed by range predicates over inputs; our framework augments the generic GGM construction framework of [17] to provide two concrete DPRF schemes, namely BRC and URC.
- We prove the security of our constructions in a modular way, thus also yielding the first security analysis of similar GGM-based key-delegation schemes, and the union-policy privacy of URC.
- We describe several key applications of DPRFs in the context of efficient key-delegation protocols for authentication, access control and encryption purposes.

Paper organization. Section 2 reviews related work. Section 3 formulates the DPRF primitive, Section 4 presents our two DPRF constructions for the case of range policies. Section 5 elaborates on the applicability of DPRFs. Finally, Section 6 concludes our paper with directions to future work. Selected proofs appear in the Appendix.

2 Related Work

We start by reviewing existing work related to DPRFs.

Secure delegation of computations. The notion of delegation of cryptographic operations is already mature: Starting from early work on proxy signatures [28] and proxy cryptography [5], basic primitives such as signatures (e.g., [6, 28]) and encryption (e.g.,[2, 19, 22]) have been studied in the context of an untrusted proxy who is authorized to operate on signatures or ciphertexts. Recently, there has also been an increased interest in verifiability and privacy of general outsourced computations (e.g., [1, 9, 10, 18, 33]) or specific crypto-related operations (e.g., [4, 15, 20]). To the best of our knowledge, however, no prior work explicitly and formally examines the delegation of PRFs.[1]

PRF extensions. Closer to our new DPRF primitive are the known extensions of PRFs, namely the *verifiable PRFs* (VPRFs) (e.g., [12, 27, 29]) and the *oblivious PRFs* (OPRFs) (e.g., [16, 23]). A VPRF provides a PRF value along with a non-interactive proof, which enables anyone to verify the correctness of the PRF value. Although such proofs can be useful in third-party settings, they are not related to the delegation of the PRF evaluation without the secret key. Similarly, an OPRF is a two-party protocol that securely implements functionality $(k, x) \rightarrow (\perp, f_k(x))$—that is, a party evaluates a PRF value without knowledge of the secret

[1] Delegating PRF evaluation in a similar vein as we consider here, was considered in [7, 8] after our work was submitted for publication.

key. Yet, although the party that provides the key can be viewed to preserve resources, this setting does not match ours because there is no (i) input privacy (it is the second party who provides the input x), and (ii) bandwidth efficiency when applied over many values. Other related work includes *algebraic PRFs*, employed in [4] to achieve optimal private verification of outsourced computation.

GGM framework. This refers to the seminal work by Goldreich *et al.* [17], which shows how to generically construct a PRF given black-box access to a length-doubling PRG. The approach is based on a tree structure, over which hierarchical invocations of the PRG produce the PRF values at the leaves. Our constructions extend this framework in non-trivial ways: First, we support delegation of PRF evaluations; second, our security is proved in a much stronger adversarial setting where the adversary gets to adaptively learn also *internal* PRG values. The GGM framework, along with its tree-based key-derivation structure, have been widely used in the literature; also for special/limited delegation purposes in the context of access control [30] and forward security [21]. Yet, to the best of our knowledge, no such known key-derivation method has been analyzed fully in a security model like ours, which allows for adaptive PRG queries and addresses policy privacy issues.

3 Definitions

A *pseudorandom function family (PRF)* $\mathcal{F}$ is a family of functions $\{f_k : A \to B \mid k \in \mathcal{K}\}$ so that $\mathcal{K}$ is efficiently samplable and all $\mathcal{F}, \mathcal{K}, A, B$ are indexed by a security parameter λ. The security property of a PRF is as follows: For any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ running in time polynomial in λ it holds that

$$|\Pr[\mathcal{A}^{f_k(\cdot)} = 1] - \Pr[\mathcal{A}^{R(\cdot)} = 1]| = \mathrm{negl}(\lambda)\ ,$$

where negl denotes a negligible function and the probability above is taken over the coins of $\mathcal{A}$ and the random variables k and R which are uniform over the domains $\mathcal{K}$ and $(A \to B)$ respectively.

Delegatable PRFs. The notion of delegation for PRFs is defined with respect to a *delegation policy* $\mathcal{P}$, i.e., $\mathcal{P}$ is a set of predicates defined over A, also indexed by λ, where each predicate P has an efficient encoding; the set of elements in A that satisfy P is denoted as $A_P = \{x \in A \mid P(x)\}$.

Definition 1 (Delegatable PRF) *A triple $(\mathcal{F}, T, C)$ is a* delegatable PRF (DPRF) w.r.t. policy $\mathcal{P}$ *provided it satisfies two properties, correctness and security, that are defined individually below.*

Correctness. T is a PPT algorithm such that, given a description of $P \in \mathcal{P}$ and a key $k \in \mathcal{K}$, it outputs a "trapdoor" τ. The latter is intended to be used along with the deterministic polynomial-time algorithm C for the computation of every element of A that satisfies the predicate P. For fixed P, k, the algorithm C can be considered as a function

$$C : St_{P,k} \longrightarrow B \times St_{P,k}\ ,$$

where $St_{P,k}$ is a set of states, and the output $C(s)$ is a pair that consists of a PRF value and a (new) state. We denote $C(s) = \langle C_L(s), C_R(s)\rangle$ and define recursively the set of *reachable* states from a subset S of $St_{P,k}$ as

$$\mathcal{R}(S) \triangleq S \cup \mathcal{R}(C_R(S)).$$

The elements of the DPRF that are produced given an initial state s will be defined using the complete set of reachable states given s. For a singleton $S = \{s\}$, we will write $\mathcal{R}(s)$ instead of $\mathcal{R}(\{s\})$, and we will denote by $\mathcal{R}(s)$ the closure of s under $\mathcal{R}$, i.e., the fixpoint of the recursive equation for $\mathcal{R}$ that also contains s.

Definition 2 (Correctness) *The DPRF scheme* $(\mathcal{F}, T, C)$ *is* correct *for a policy* $\mathcal{P}$ *if for every* $P \in \mathcal{P}$*:*

1. $\{\tau \mid \tau \leftarrow T(P,k)\} \cup \{\perp\} \subseteq St_{P,k}$*, for any* k*.*
2. $C_R(\perp) = \perp$ *(*termination condition*).*
3. *There is a polynomial* q *s.t. for every* $k, \tau \leftarrow T(P,k)$*:*
   - *(i)* $\perp \in \mathcal{R}(\tau)$ *(*termination guarantee*).*
   - *(ii)* $|\mathcal{R}(\tau)| \leq q(\lambda)$ *(*feasibility*).*
   - *(iii)* $\{f_k(x) \mid P(x)\} = B_P = \{C_L(s) \mid s \in \mathcal{R}(\tau)\}$ *(*completeness*).*

According to the above conditions, all possible trapdoors corresponding to a certain policy predicate P are valid initial inputs for C. Starting from an arbitrary trapdoor for P, the proxy can execute a number of steps, compute the DPRF image of every argument x that fulfills P, and terminate when it reaches the final state $\perp$, where no further useful information can be derived.

We note that condition 3.(ii) implies the restriction that the size of every policy predicate is polynomial. This stems from the fact that we consider the setting where the proxy wishes to eventually compute *all* the delegated PRF values. If this is not necessary (or desirable) for the DPRF application, the condition can be relaxed to any size of A_P (including super-polynomial sizes). In this case, completeness (item 3.(iii) above) will not suffice since the proxy cannot hope to be able to compute all the delegated PRF values. There are a number of ways to capture this by suitably modifying the way C works; for instance: (i) C may sample the uniform distribution over B_P, (ii) C may be given the value x as input, and return $f_k(x)$ if $x \in A_P$ and $\perp$ otherwise, (iii) C may be given the lexicographic rank of an element x within A_P and return $f_k(x)$ or $\perp$ otherwise.

Security. For security we consider the case where the server is malicious and model DPRF security as a game $\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}$ between an attacker $\mathcal{A}$ and a challenger $\mathcal{C}$ indexed by parameter λ. Due to the delegation capabilities of DPRFs, the security game is more elaborate than the security of a plain PRF, as shown next.

DPRF Security Game $\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda)$

1. The challenger $\mathcal{C}$ selects k from $\mathcal{K}$.
2. The adversary $\mathcal{A}$ is allowed to interact with $\mathcal{C}$ and ask two types of queries:
   (a) *PRF queries* for a value $x \in A$; to those queries $\mathcal{C}$ responds with $f_k(x)$ and adds the value x to a set L_{que}.
   (b) *delegation queries* for a policy predicate $P \in \mathcal{P}$; to those queries $\mathcal{C}$ responds with $\tau \leftarrow T(P,k)$ and adds P to a set L_{pol}.
3. The adversary $\mathcal{A}$ submits a challenge query x^* to which the challenger $\mathcal{C}$ responds as follows: it flips a coin b and if $b = 1$ it responds with $y^* = f_k(x^*)$, otherwise responds with a random value y^* from B.
4. The adversary $\mathcal{A}$ continues as in step 2.
5. The adversary $\mathcal{A}$ terminates by returning a single bit $\tilde{b}$. Subsequently the game returns a bit which is 1 if and only if the following holds true:

$$(b = \tilde{b}) \wedge (x^* \notin L_{\mathsf{que}}) \wedge \forall P \in L_{\mathsf{pol}} : \neg P(x^*).$$

Definition 3 (Security) *A DPRF scheme $(\mathcal{F}, T, C)$ is* secure *for a policy $\mathcal{P}$ if for any PPT $\mathcal{A}$, it holds that*

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

We make the following observations about the definition. First, it is easy to see that a delegatable PRF is indeed a PRF. Specifically, any PRF attacker $\mathcal{A}$ against f_k can be turned into an attacker $\mathcal{A}'$ that wins the DPRF game described above. We provide only a simple sketch of this which follows by a standard "walking" argument (the reader familiar with such arguments may skip to the next paragraph). Fix some PPT $\mathcal{A}$ and let α be its non-negligible distinguishing advantage. There will be some polynomial q so that, for any λ, $q(\lambda)$ is an upper bound on the number of queries submitted to $\mathcal{A}$'s oracle by $\mathcal{A}$. Given such q and for fixed λ, we define the *hybrid* oracle $(f_k/R)_j$ for any $j \in \{0, \ldots, q(\lambda)\}$ that operates as follows: $(f_k/R)_j$ responds as $f_k(\cdot)$ in the first j queries and as $R(\cdot)$ in the last $q(\lambda) - j$ queries. Taking such sequence of oracles into account, by triangular inequality, it follows that there exists some value $j \in \{0, \ldots, q(\lambda) - 1\}$ for which the distinguishing probability will be at least $\alpha/q(\lambda)$ for $\mathcal{A}$ to distinguish between two successive hybrid oracles $(f_k/R)_j$ and $(f_k/R)_{j+1}$ when R is a random function. This follows from the fact that $\mathcal{A}$ distinguishes the "extreme" hybrids $R(\cdot)$ and $f_k(\cdot)$ with probability α. We now construct $\mathcal{A}'$ as follows out of $\mathcal{A}$: $\mathcal{A}'$ plays the DPRF game and queries the DPRF function for the first j queries of $\mathcal{A}$. Then it submits the $(j+1)$-th query of $\mathcal{A}$ as the challenge. Finally, it completes the simulation of $\mathcal{A}$ by answering any remaining queries of $\mathcal{A}$ with random values drawn from B (w.l.o.g. the queries to the oracle are all distinct). It is easy to see that the distinguishing advantage of $\mathcal{A}'$ is $\alpha/q(\lambda)$, i.e., non-negligible in λ.

Second, we observe that there is a trivial construction of a delegatable PRF from any PRF: Consider an ordering $\leq$ over A, e.g., the lexicographical order. For fixed P, k set $T(P, k) = \langle f_k(x_1), \ldots, f_k(x_{|A_P|})\rangle = \tau$, where x_i is the i-th element of A_P according to $\leq$. Given τ, the set of states is $St_{P,k} = \{\tau, (2, \tau), \ldots, (|A_P|, \tau), \perp\}$ and the reconstruction function C can be simply defined to be a table-lookup. It is straightforward to show that $(\mathcal{F}, T, C)$ is a DPRF as long as the underlying family $\mathcal{F}$ is a PRF, since any delegation query can be interpreted as a series of polynomially many PRF queries.

The existence of a trivial DPRF construction w.r.t. arbitrary policies from any given PRF motivates our primitive: Interesting DPRF constructions will be those that are *bandwidth efficient*, i.e., they provide trapdoors with size that is sublinear in the number of elements that satisfy the corresponding policy predicate.

Policy privacy. We next introduce a property that goes beyond the standard (D)PRF security and is relevant in the context of a delegatable PRF. In order to model this privacy condition we use an indistinguishability game $\mathcal{G}^{\mathcal{A}}_{\mathsf{PP}}$ carried out between an attacker $\mathcal{A}$ and a challenger $\mathcal{C}$ indexed by a security parameter λ. The game proceeds as follows:

Definition 4 (Policy privacy) *A DPRF scheme $(\mathcal{F}, T, C)$ for a policy $\mathcal{P}$ satisfies* policy privacy *if for any PPT $\mathcal{A}$, it holds that*

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{PP}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

The above definition suggests that the trapdoor that corresponds to a certain policy predicate hides the predicate itself (i) at least among all policy predicates that enable the same number of elements, and (ii) when the adversary does not make queries whose responses leak unequal parts of the PRF image of the two challenge predicates. Observe that all the restrictions stated are necessary: if $|A_{P_0}| \neq |A_{P_1}|$, then the adversary can distinguish P_0 from P_1 by counting the number of new PRF values it computes starting from state τ^* and ending in $\perp$. In addition, if the number of elements that satisfy simultaneously any set S of

DPRF Policy Privacy Security Game $\mathcal{G}^{\mathcal{A}}_{\mathsf{PP}}(1^\lambda)$

1. The challenger $\mathcal{C}$ selects k from $\mathcal{K}$.
2. The adversary $\mathcal{A}$ is allowed to interact with $\mathcal{C}$ and ask delegation queries $P \in \mathcal{P}$.
3. The adversary $\mathcal{A}$ submits two policy predicates P_0, P_1 to $\mathcal{C}$. The challenger flips a bit b and computes $\tau^* \leftarrow T(P_b, k)$. It returns τ^* to the adversary.
4. The adversary continues as in step 2 and terminates returning a bit $\tilde{b}$. The game terminates with 1 provided that

$$(b = \tilde{b}) \wedge (|A_{P_0}| = |A_{P_1}|) \wedge (A_{P_0} \neq A_{P_1}) \wedge$$
$$\wedge \forall S \subseteq L_{\mathsf{pol}} : \Big|(\bigcap_{P \in S} A_P) \cap A_{P_0}\Big| = \Big|(\bigcap_{P \in S} A_P) \cap A_{P_1}\Big|.$$

delegation queries and P_0 is different than the number of elements that satisfy S and P_1, then the adversary can distinguish the two predicates by counting the size of $\{C_L(s) \mid s \in \mathcal{R}(\tau^*)\} \cap (\bigcap_{P \in S} B_P)$.

Multiple-challenge policy privacy. While the above formulation can capture a wide set of attacks against privacy, it can be strengthened further by allowing *multiple* challenge pairs of policy predicates. In this case, the security game allows a multiple number of adversarial actions submitting pairs of policies (as in step 3 of the game) to be interleaved with delegation queries. The challenger always responds based on the same coin flip b. Interestingly, this *multiple-challenge policy privacy* formulation can be shown to be strictly stronger than Definition 4.

We next argue that even though desirable, the above formulations of privacy conflict with our efficiency considerations (sublinear-size trapdoors) for a wide class of schemes. This will motivate relaxing the policy privacy property as we will see in the end of the section; the reader unwilling to go over the details of the lower bound type of argument we sketch below may skip directly to the policy privacy relaxation in the end of the section.

Assume that the policy predicates are ranges $[a, b]$ that lie in an interval $[0, 2^\lambda - 1]$. Then, no efficient and policy private DPRF scheme exists, if the trapdoor generation algorithm $T(P, k)$ is deterministic, and the delegated computation is *tree-wise*, i.e., for each range, the trapdoor is a data structure of keys and meta-data that enable in a deterministic way the calculation of the final set of values through a tree-like deterministic derivation whose generated tree structure depends only on the meta-data. Indeed, if policy privacy is satisfied, then for every $0 < j \leq b - a$, the delegated computation of the intersection $[a + j, b]$ PRF values must be *indistinguishable* for $[a, b]$ and $[a + j, b + j]$. Otherwise, an adversary can make queries for all the PRF values in $[a + j, b]$ and, since it knows the way that each unique trapdoor of $[a, b]$ and $[a + j, b + j]$ computes the PRF values of the range $[a + j, b]$, can thus distinguish the two range predicates by making the corresponding equality tests. By induction, this implies that for any $j \in \{0, \ldots, b - a\}$, in the trapdoor of $[a, b]$ there exist distinct keys $d_0, \ldots, d_j$ that allow the computation of $f_k(a), \ldots, f_k(a + j)$, respectively. Thus, the size of the trapdoor of the range $[a, b]$ consists of at least $r = b - a + 1 = \#[a, b]$ keys which means that the DPRF cannot be efficient (i.e., enabling the delegation of the range with a trapdoor size less than the set of values being delegated).

The above argument suggests that, if policy privacy is to be attained, we need trapdoors at least as long as the values we delegate. Note the trivial construction for ranges mentioned above, i.e., when the trapdoor of $[a, b]$ is the $\langle f_k(a), f_k(a+1), \ldots, f_k(b)\rangle$, does not satisfy policy privacy. For instance, when delegating $[1, 3]$ and $[3, 5]$ the attacker can easily distinguish them by obtaining the value $f_k(3)$ (it belongs in the intersection hence the attacker is allowed to have it) and checking its location within the trapdoor vector. To achieve

policy privacy for the trivial construction one needs to additionally permute the PRF values in the trapdoor in some fashion: it can be easily shown that a lexicographic ordering of these values suffices.

In our upcoming constructions, we obtain efficient DPRFs that use tree-wise derivations where the values computed are at the leaves of a full binary tree, and the policy predicates are ranges covered by proper subtrees. In this case, even allowing a probabilistic trapdoor generator does not suffice to provide policy privacy for a very efficient scheme. To see this, let $[a, b]$ be a range of size r and τ be a trapdoor for $[a, b]$ of size $g(r) = O(\log^c r)$, i.e., $g(r)$ is polylogarithmic in r, hence also in λ. By the Pigeonhole principle, there exists a delegation key y in τ that computes all the values corresponding to a set $T \subseteq [a, b]$ of size at least $r/g(r)$, that is, a subtree T of depth at least $d \geq \log(r) - \log(g(r)) = \omega(1)$. Assuming w.l.o.g. that $r/g(r)$ is an integer, the latter implies that there exists a value $x \in [a, b]$ with an *all-one suffix* of length d that is computed by this key y. Since there are $g(r)$ such possible values $x \in [a, b]$, an adversary has significant probability $1/g(r)$ of guessing x. Then it can make a query x, receive $f_k(x)$ and submit as the challenge the two policies $P_0 = [a, b]$ and $P_1 = [x, b + (x - a)]$. After it receives the challenge trapdoor τ^*, it can locate all keys that correspond to subtrees of size $\geq d$ and check if there is a minimum leaf value (an all-zero path) in some of these subtrees that equals to $f_k(x)$. Then, the adversary can distinguish effectively P_0, P_1 since x cannot be covered by any subtree of size d in a trapdoor for $[x, b + (x - a)]$ (it must be covered by a leaf while $d > 1$). This argument can be extended to more general tree-like delegation schemes and for the case $g(r) = o(r)$ but we omit further details as it already demonstrates that efficient tree-like constructions require a somewhat more relaxed definition of policy privacy (which we introduce below).

Union policy privacy. We finally introduce the notion of *union policy privacy*, where the adversary is restricted from making queries in the *union* of the challenge policy predicates (but is allowed to query at arbitrary locations outside the targeted policy set). This is a strict relaxation of Definition 4, and we model it by a game $\mathcal{G}^{\mathcal{A}}_{\mathsf{UPP}}(1^\lambda)$ that proceeds *identically* as $\mathcal{G}^{\mathcal{A}}_{\mathsf{PP}}(1^\lambda)$, but terminates with 1 provided that all the following (weaker set of) conditions are met: $b = \tilde{b}$, $|A_{P_0}| = |A_{P_1}|$, $A_{P_0} \neq A_{P_1}$ and $\forall P \in L_{\mathsf{pol}}$: $A_P \cap (A_{P_0} \cup A_{P_1}) = \emptyset$. To see the connection with Definition 4 observe that the latter condition is equivalent to

$$\forall S \subseteq L_{\mathsf{pol}} : \Big|\big(\bigcap_{P \in S} A_P\big) \cap A_{P_0}\Big| = \Big|\big(\bigcap_{P \in S} A_P\big) \cap A_{P_1}\Big| = 0.$$

Note that for policies $\mathcal{P}$ consisting of disjoint predicates, the games $\mathcal{G}^{\mathcal{A}}_{\mathsf{PP}}(1^\lambda)$ and $\mathcal{G}^{\mathcal{A}}_{\mathsf{UPP}}(1^\lambda)$ are equivalent.

4 Constructions

In this section we present DPRF schemes for *range* policy predicates. In Section 4.1 we describe a first construction, called *best range cover* (BRC), which satisfies the correctness and security properties of DPRFs, achieving trapdoor size logarithmic in the range size. However, BRC lacks the policy privacy property. In Section 4.2 we build upon BRC and obtain a (union) policy-private DPRF scheme, called *uniform range cover* (URC), which retains the trapdoor size complexity of BRC. In Section 4.3 we include the security proofs of the two schemes. In Section 4.4 we prove the union policy privacy property of URC.

4.1 The BRC Construction

Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a pseudorandom generator and $G_0(k), G_1(k)$ be the first and second half of the string $G(k)$, where the specification of G is public and k is a secret random seed. The GGM pseudorandom function family [17] is defined as $\mathcal{F} = \{f_k : \{0,1\}^n \to \{0,1\}^\lambda\}_{k \in \{0,1\}^\lambda}$, such that

$f_k(x_{n-1} \cdots x_0) = G_{x_0}(\cdots(G_{x_{n-1}}(k)))$, where n is polynomial in λ and $x_{n-1} \cdots x_0$ is the input bitstring of size n.

The GGM construction defines a binary tree on the PRF domain. We illustrate this using Figure 2, which depicts a binary tree with 4 levels. The leaves are labeled with a decimal number from 0 to 15, sorted in ascending order. Every edge is labeled with 0 (resp. 1) if it connects a left (resp. right) child. We label every *internal* node with the binary string determined by the labels of the edges along the path from the root to this node. Suppose that the PRF domain is $\{0,1\}^4$. Then, the PRF value of 0010 is $f_k(0010) = G_0(G_1(G_0(G_0(k))))$. Observe that the composition of G is performed according to the edge labels in the path from the root to leaf $2 = (0010)_2$, selecting the first (second) half of the output of G when the label of the visited edge is 0 (resp. 1). Based on the above, the n-long binary representation of the leaf labels constitute the PRF domain, and every leaf is associated with the PRF value of its label.

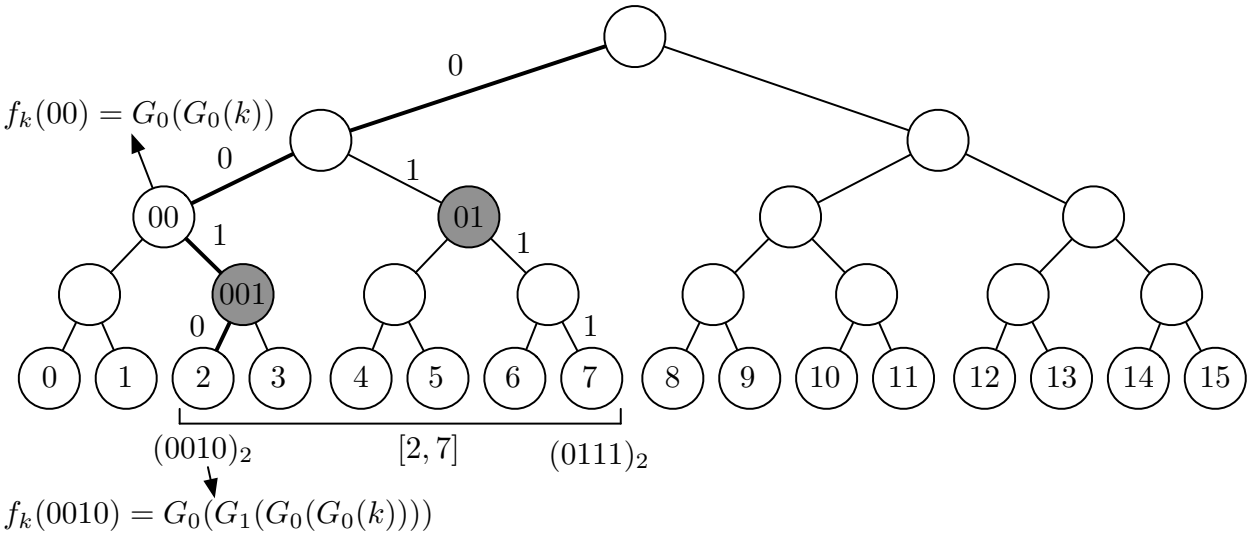


Figure 2: A GGM tree example

Note that we can also associate every *internal* node of the GGM tree with a *partial* PRF value, by performing the composition of G as determined by the path from the root to that node. For example, node 00 in Figure 2 is associated with partial PRF $G_0(G_0(k))$. Henceforth, for simplicity, we denote by $f_k(x_{n-1} \cdots x_j)$ the partial PRF $G_{x_j}(\cdots(G_{x_{n-1}}(k)))$. Observe that if a party has the partial PRF $f_k(x_{n-1} \cdots x_j)$, then it can compute the PRF values of all 2^j inputs that have prefix $x_{n-1} \cdots x_j$, simply by following a DFS traversal in the subtree with root $x_{n-1} \cdots x_j$ and composing with seed $f_k(x_{n-1} \cdots x_j)$. In our running example, using the partial PRF value at node 00, we can derive the PRF values of the inputs in (decimal) range $[0,3]$ as $f_k(0000) = G_0(G_0(f_k(00)))$, $f_k(0001) = G_1(G_0(f_k(00)))$, $f_k(0010) = G_0(G_1(f_k(00)))$, and $f_k(0011) = G_1(G_1(f_k(00)))$.

For any range $[a,b]$ of leaf labels, there is a (non-unique) set of subtrees in the GGM tree that *cover* exactly the corresponding leaves. For instance, $[2,7]$ is covered by the subtrees rooted at nodes 001 and 01 (colored in grey). According to our discussion above, a party having the partial PRF values of these subtree roots and the subtree depths, it can derive all the PRF values of the leaves with labels in $[a,b]$. In our example, having $(f_k(001),1)$ and $(f_k(01),2)$, it can derive the PRF values of the leaves with labels in $[2,7]$. Our first construction is based on the above observations. In particular, given a range policy predicate $[a,b] \in \mathcal{P}$ with size $|A_P| = b - a + 1$, it finds the *minimum* number of subtrees that cover $[a,b]$. As such,

we call this scheme as *best range cover* (BRC).

The BRC DPRF construction is a triplet $(\mathcal{F}, T, C)$, where $\mathcal{F}$ is the GGM PRF family described above with tree depth n. The delegation policy is $\mathcal{P} = \{[a,b] \mid 0 \leq a \leq b \leq a + \lambda^\gamma < 2^n\}$, where γ is a constant integer. Note that when $a = b$ the trapdoor sent to the proxy is simply the PRF value $f_k(a)$. In the rest of the section, we focus on the non-trivial case where the range is not a singleton. When $a < b$, the trapdoor generation algorithm T of BRC is given below. This algorithm takes as input a secret key k and a range predicate $[a,b] \in \mathcal{P}$. It outputs a delegation trapdoor τ that enables the computation of $f_k(x)$ for every x whose decimal representation is in $[a,b]$ (i.e., the PRF values of the leaves in the GGM tree with labels in $[a,b]$). T initially finds the first bit in which a and b differ (Line 2), which determines the common path from the root to leaves a and b. Suppose that this common path ends at node u. If $[a,b]$ is the set of labels that belong to the subtree with root u, then T outputs a single pair consisting of the partial PRF value that corresponds to u along with the depth of the subtree (Lines 3-5). Otherwise, T traverses the left and right subtree of u separately (Lines 3,7-13 and 14-21, respectively). We will describe only the left traversal (the right one is performed symmetrically). T considers the path $p_{v \to a}$ starting from the left child of u, denoted by v, to a. It checks whether a is the leftmost leaf of v's subtree. In this case the PRF value of v is included in τ along with the depth of v's subtree, and the traversal terminates (Lines 3,7). Otherwise, if $p_{v \to a}$ proceeds to the left child of v, it includes in the trapdoor the PRF value of the right child of v together with the depth of the subtree rooted at that child. In any case, it continues the traversal in the same fashion with the next node of $p_{v \to a}$ (Lines 8-13).

The Trapdoor Generation Algorithm T in BRC

Input: $a, b : 0 \leq a < b \leq a + \lambda^\gamma \leq 2^n - 1$ and $k \in \{0,1\}^\lambda$

Output: Trapdoor τ for computing the PRFs for $[a,b]$

1. $\tau \leftarrow \langle\rangle$
2. $t \leftarrow \max\{i \mid a_i \neq b_i\}$
3. **if** $(\forall i \leq t : a_i = 0)$ **then**
4. **if** $(\forall i \leq t : b_i = 1)$ **then**
5. **return** $(f_k(a_{n-1} \cdots a_{t+1}), t+1)$
6. **else**
7. Append $(f_k(a_{n-1} \cdots a_t), t)$ to τ
8. **else**
9. $\mu \leftarrow \min\{i \mid i < t \wedge a_i = 1\}$
10. **for** $i = t-1$ to $\mu + 1$
11. **if** $a_i = 0$ **then**
12. Append $(f_k(a_{n-1} \cdots a_{i+1}1), i)$ to τ
13. Append $(f_k(a_{n-1} \cdots a_\mu), \mu)$ to τ
14. **if** $(\forall i \leq t : b_i = 1)$ **then**
15. Append $(f_k(b_{\nu-1} \cdots b_t), t)$ to τ
16. **else**
17. $\nu \leftarrow \min\{i \mid i < t \wedge b_i = 0\}$
18. **for** $i = t-1$ to $\nu + 1$
19. **if** $b_i = 1$ **then**
20. Append $(f_k(b_{n-1} \cdots b_{i+1}0), i)$ to τ
21. Append $(f_k(b_{n-1} \cdots b_\nu), \nu)$ to τ
22. **return** τ

In the example of Figure 2, for input $[2,7]$, T outputs trapdoor $\tau = \langle (f_k(001), 1), (f_k(01), 2) \rangle$. By its description, it is easy to see that the algorithm covers the input range with *maximal* subtrees.

We next describe the reconstruction algorithm C of BRC. For fixed $P = [a, b]$ of size r and key k, we define the set of states as $St_{P,k} = \{\tau, (1, \tau), \ldots, (r-1, \tau), \perp\}$, where $\tau = \langle (y_1, d_1), \ldots, (y_m, d_m)\rangle$ is the trapdoor produced by algorithm T. Note that, every $y_i, i = 1, \ldots, m$ corresponds to a partial PRF value associated with the root of a GGM subtree. Therefore, there is a natural ordering of PRF values for a given τ, starting from the leftmost leaf of the subtree of y_1 to the rightmost leaf of the subtree of y_m. This order is not necessarily the same as the order of the leaves in the GGM tree. Starting from the PRF value of the leftmost leaf of the y_1 subtree, C computes in every next step the PRF value of the next leaf in the ordering discussed above. Specifically, C starts with state τ and computes $C(\tau) = \langle G_0(\cdots(G_0(y_1))), (1, \tau)\rangle$, where the G_0 composition is performed d_1 times. Next, given state (i, τ), C locates the unique subtree that covers the $(i+1)$-th leaf x in the ordering of τ. Specifically, it finds the pair (y_t, d_t) in τ, where t is such that $\sum_{\rho=1}^{t-1} 2^{d_\rho} \leq i < \sum_{\rho=1}^{t} 2^{d_\rho}$. Then, the suffix $x_{d_t} \cdots x_0$ is the binary representation of $i - \sum_{\rho=1}^{t-1} 2^{d_\rho} \in [0, 2^{d_t} - 1]$. Given this suffix, C can compute $f_k(x) = G_{x_0}(\cdots(G_{x_{d_t}}(y_t)))$ and output $C((i, \tau)) = \langle G_{x_0}(\cdots(G_{x_{d_t}}(y_t))), (i+1, \tau)\rangle$. Note that, for input state $(r-1, \tau)$, C outputs $\langle G_1(\cdots(G_1(y_m))), \perp\rangle$, where the G_1 composition is performed d_m times.

Considering the tree of Figure 2, the trapdoor for range $[2, 14]$ is $\langle (f_k(01), 2), (f_k(001), 1), (f_k(10), 2), (f_k(110), 1), (f_k(1110), 0)\rangle$. Algorithm C computes the PRF values for leaves 4, 5, 6, 7, 2, 3, 8, 9, 10, 11, 12, 13, 14 in this order, i.e.,

$$
\begin{aligned}
&C(\tau) = \langle G_0(G_0(f_k(01))), (1, \tau)\rangle = \langle f_k(0100), (1, \tau)\rangle \rightarrow \\
&C((1, \tau)) = \langle G_1(G_0(f_k(01))), (2, \tau)\rangle = \langle f_k(0101), (2, \tau)\rangle \rightarrow \\
&\vdots \\
&C((12, \tau)) = \langle f_k(1110), \perp\rangle.
\end{aligned}
$$

Based on the above description, some partial PRF values are computed *multiple* times in C, e.g., $f_k(011)$ is computed twice; once during calculating $f_k(0110)$ and once for $f_k(0111)$. Note that this can be easily avoided by employing a cache of size $O(r)$; every PRF value is computed once, stored in the cache, and retrieved in a next step if necessary. In this manner, the computational cost of C can become $O(r)$, since we compute *one* PRF value for every node in the subtrees covering the range (of size r).

The correctness of BRC is stated in the following theorem.

Theorem 1 *The BRC DPRF construction $(\mathcal{F}, T, C)$ w.r.t. policy $\mathcal{P} = \{[a, b] \mid 0 \leq a \leq b \leq a + \lambda^\gamma \leq 2^n - 1\}$, where n is the depth of the underlying GGM tree, γ is a constant integer, and λ^γ is the maximum range size, is correct.*

Proof. We will prove that all conditions of Definition 2 are satisfied. Condition 1 can be easily seen to be satisfied by the definition of the set of states $St_{P,k}$. Moreover, since $\mathcal{R}(\tau) = St_{P,k}$, conditions 2, 3.(i) and 3.(ii) are directly satisfied by the description of algorithm C and because the size of range r (which determines the number of values conforming to the range policy predicate) is upper bounded by λ^γ which is a polynomial. Therefore, it remains to prove that condition 3.(iii) is satisfied. Based on our description on C, it is apparent that the algorithm computes exactly the PRF values of the leaves covered by the subtrees corresponding to the partial PRF values included in τ by algorithm T. Hence, it suffices prove that the labels of these leaves are *exactly* $[a, b]$.

Let $t = \max\{i \mid a_i \neq b_i\}$. Also, let V_1, V_2 be the sets of arguments of the PRF values that are computed in Lines 3-13 and 14-21 of algorithm T, respectively. If $V_2 = \emptyset$, then the input range is exactly covered by

a single subtree (checks in lines 3 and 4 are true) and $V_1 = [a, b]$. Otherwise, it holds that

$$\begin{aligned} V_1 &= \{x \in [0, 2^n - 1] \mid x_{n-1} \cdots x_t = a_{n-1} \cdots a_t \wedge \\ &\quad \wedge (\exists i < t : [x_i = 1 \wedge a_i = 0] \vee x = a)\} = \\ &= \{x \in [0, 2^n - 1] \mid a \leq x \leq a_{n-1} \cdots a_t 1 \cdots 1\} \\ &= [a, a_{n-1} \cdots a_t 1 \cdots 1] \quad . \end{aligned}$$

Similarly, we get that $V_2 = [b_{n-1} \cdots b_t 0 \cdots 0, b]$. Observe that, by the definition of t, $a_t = 0$ and $b_t = 1$. Thus, it holds that $a_{n-1} \cdots a_t 1 \cdots 1 + 1 = b_{n-1} \cdots b_t 0 \cdots 0$. This means that $[a, b] = V_1 \cup V_2$, which concludes our proof.

We next discuss the trapdoor size complexity in BRC. Let V_1, V_2 defined as previously. Then, $|V_1| + |V_2| = r$. We will analyze $|V_1|$ (the analysis for V_2 is similar). Observe that, in every step in Lines 3-13, the algorithm covers more than $\lfloor |V_1|/2 \rfloor$ values of V_1 with a *maximal* subtree of the sub-range defined by V_1. Iteratively, this means that the algorithm needs no more than $\log(r)$ maximal subtrees to cover the entire sub-range of V_1. Consequently, the total number of elements in τ is $O(\log(r))$.

We have explained the correctness of BRC and its efficient trapdoor size. We also prove its security in Section 4.3. However, BRC does not satisfy policy privacy, even for non-intersecting policy predicates. We illustrate with a simple example using the tree from Figure 2. Consider ranges $[2, 7]$ and $[9, 14]$, both with size 6. The trapdoors generated for these ranges are $\langle((f_k(001), 1), (f_k(01), 2)\rangle$ and $\langle(f_k(101), 1), (f_k(1001), 0), (f_k(110), 1), (f_k(1110), 0)\rangle$, respectively. Clearly, these trapdoors are distinguishable due to their different sizes. This motivates our second DPRF construction presented in the next sub section.

4.2 The URC Construction

Consider again the ranges $[2, 7]$ and $[9, 14]$ in the tree of Figure 2, for which BRC generates two distinguishable trapdoors, $\langle((f_k(001), 1), (f_k(01), 2)\rangle$ and $\langle(f_k(101), 1), (f_k(1001), 0), (f_k(110), 1), (f_k(1110), 0)\rangle$, respectively. Instead of computing the trapdoor of $[2, 7]$ as above, assume that we generate an alternative trapdoor equal to $\langle(f_k(010), 1), (f_k(0010), 0), (f_k(011), 1), (f_k(0011), 0)\rangle$. Observe that this trapdoor appears to be indistinguishable to that of $[9, 14]$; indeed, the two trapdoors have the same number of elements, the first parts of their elements are all partial PRFs, whereas their second parts (i.e., the depths) are pairwise equal. This suggests that, we could achieve policy privacy, if we could devise a trapdoor algorithm T such that, for *any* range predicate of a fixed size r it *always* generates a trapdoor with a *fixed* number of elements and a *fixed* sequence of depths. More simply stated, the algorithm should produce *uniform* trapdoors for ranges of the same size. The challenge is to design such an algorithm retaining the logarithmic trapdoor size of BRC. Next, we present our second DPRF construction, called *uniform range cover* (URC), which enjoys the efficiency of BRC and the union policy privacy property.

URC builds upon BRC. In particular, it starts by producing a trapdoor as in BRC, and then modifies it to generate a uniform trapdoor for the given range r. Before embarking on its detailed description, we must investigate some interesting properties of the trapdoors of BRC, and provide some important definitions. Recall that a trapdoor in BRC is a sequence of elements, where the first part is a (full or partial) PRF value, and the second is a depth value.

Definition 5 *Let r be an integer greater than 1. A pair of non-negative integral sequences $D = ((k_1, \ldots, k_c), (l_1, \ldots, l_d))$ is called a* decomposition *of r if the following hold:*

(i) $\sum_{i=1}^{c} 2^{k_i} + \sum_{j=1}^{d} 2^{l_j} = r$

(ii) $k_1 > \cdots > k_c$ *and* $l_1 > \cdots > l_d$

A decomposition of r $D = ((k_1, \ldots, k_c), (l_1, \ldots, l_d))$ *is a* worst-case decomposition *(w.c.d.) of* r *if it is of maximum size, i.e., for every decomposition of* r $D' = ((k'_1, \ldots, k'_{c'}), (l'_1, \ldots, l'_{d'}))$*, we have that* $c' + d' \leq c + d$*. We define* $M_D \triangleq \max\{k_1, l_1\}$ *as the maximum integer that appears in* D*.*

By the description of algorithm T in BRC for fixed range size r, the depths in the trapdoor can be separated into two sequences that form a decomposition of r, unless the input range is exactly covered by a single subtree. Each sequence corresponds to a set of full binary subtrees of *decreasing* size that cover leaves in the range predicate. The usage of the worst case decomposition will become clear soon.

The following lemma shows that the maximum integer that appears in *any* decomposition of r and, hence, the maximum depth of a subtree in a cover of a range of size r, can have just one of two consecutive values that depend only on r.

Lemma 1 *Let* $D = ((k_1, \ldots, k_c), (l_1, \ldots, l_d))$ *be a decomposition of* r*. Define* $B(r) \triangleq \lceil \log(r+2) \rceil - 2$*. Then* $M_D \in \{B(r), B(r)+1\}$*. In addition, if* $M_D = B(r)+1$ *then the second largest value is less than* M_D*.*

Proof. By the two properties of D, we have that

$$r = \sum_{i=1}^{c} 2^{k_i} + \sum_{j=1}^{d} 2^{l_j} \leq 2^{k_1+1} + 2^{l_1+1} - 2 \leq$$
$$\leq 2^{M_D+2} - 2 \Leftrightarrow 2^{M_D+2} \geq r + 2 \Rightarrow M_D \geq B(r).$$

Since $2^{B(r)+2} \geq 2^{\log(r+2)} > r \geq 2^{k_1} + 2^{l_1}$, we have that the maximum value $M_D \in \{k_1, l_1\}$ is less than $B(r)+2$ and k_1, l_1 cannot be both equal to $B(r)+1$.

By Lemma 1, the trapdoor that is generated by BRC for a range $P = [a, b]$ of size $|A_P| = r$, even if this trapdoor corresponds to a w.c.d. of r, consists of at most $|\{0, \ldots, B(r)\}| + |\{0, \ldots, B(r)+1\}| = 2B(r)+3$ pairs. Hence, the trapdoor size is $O(\log(r))$, which complies with the bound we described in Section 4.1. Moreover, since $|A_P| \leq \lambda^\gamma$, every trapdoor has no more than $2\lceil \log(\lambda^\gamma + 2) \rceil - 1$ pairs.

Observe that two trapdoors with depths that form the same decomposition appear indistinguishable. Moreover, we have proven that a trapdoor following a w.c.d. retains the logarithmic size in r. Therefore, our goal for the trapdoor algorithm T in URC is to construct a converter that takes as input a BRC trapdoor, and produces an alternative trapdoor that complies with a *fixed* w.c.d. Before proceeding to the details of T, we must prove the following vital theorem.

Theorem 2 *Let* $D = ((k_1, \ldots, k_c), (l_1, \ldots, l_d))$ *be a decomposition of* r*. Then all the elements in* $\{0, \ldots, M_D\}$ *appear in* D *iff* D *is a w.c.d. of* r*.*

Proof. Assume that the implication does not hold and let x be the maximum integer in $\{0, \ldots, M_D\}$ that does not appear in D. Since $x < M_D$, $x+1$ is in $\{0, \ldots, M_D\}$. Assume w.l.o.g. that $l_j = x+1$. If $x > k_1$, then the decomposition of r, $((x, k_1, \ldots, k_c), (l_1, \ldots, l_{j-1}, x, l_{j+1}, \ldots, l_d))$, is of greater size than that of D. If $i = \min\{i \mid x < k_i\}$, then the decomposition of r, $((k_1, \ldots, k_i, x, k_{i+1}, \ldots, k_c), (l_1, \ldots, l_{j-1}, x, l_{j+1}, \ldots, l_d))$, is of greater size than that of D. Both cases contradict to the hypothesis, hence, x must appear in D.

For the converse, let $D' = ((k'_1, \ldots, k'_{c'}), (l'_1, \ldots, l'_{d'}))$ be a w.c.d. of r. Then the integers $0, \ldots, M_{D'}$ appear in D'. By Lemma 1, all integers $0, \ldots, B(r)$ appear in D and D'. By removing the integers $0, \ldots, B(r)$ from D and D', the remaining integers are $y_1 \geq \ldots \geq y_s$ and $z_1 \geq \ldots \geq z_t$, respectively. Since an integer cannot appear more than twice in a decomposition of r and, by Lemma 1, the maximum possible value $B(r)+1$ cannot appear more than once, we have that $y_1, \ldots, y_s$ and $z_1, \ldots, z_t$ are sequences of distinct integers s.t. $\sum_{i=1}^{s} 2^{y_i} = \sum_{j=1}^{t} 2^{z_j}$. Assume that there exists a minimum index $\rho \leq s$ s.t. $y_\rho \neq z_\rho$ and that w.l.o.g $y_\rho > z_\rho$. Then we have the contradiction

$$\sum_{i=1}^{s} 2^{y_i} \geq \sum_{i<\rho} 2^{y_i} + 2^{y_\rho} > \sum_{i<\rho} 2^{y_i} + 2^{z_\rho+1} - 1 \geq$$

$$\geq \sum_{i<\rho} 2^{z_i} + \sum_{i\geq\rho} 2^{z_i} = \sum_{j=1}^{t} 2^{z_j} = \sum_{i=1}^{s} 2^{y_i}.$$

Thus, $\{y_1, \ldots, y_s\}$ and $\{z_1, \ldots, z_t\}$ are equal, and therefore D is a w.c.d. of r.

A consequence of Theorem 2 and Lemma 1 is that, for every integer $r > 1$, a w.c.d. of r is a proper rearrangement of the integers that appear in the w.c.d. of r where the first sequence is $(B(r), \ldots, 0)$ and the second sequence is the remaining integers in the decomposition in decreasing order. We term this unique w.c.d. as the *uniform decomposition* of r. The main idea in URC is to always generate a trapdoor that complies with the uniform decomposition of r.

We are ready to describe algorithm T in URC, whose pseudocode is provided below.

The Trapdoor Generation Algorithm T_{URC}

Input: $a, b : 0 \leq a < b \leq a + \lambda^\gamma \leq 2^n - 1$ and $k \in \{0,1\}^\lambda$
Output: Trapdoor τ for computing the PRFs for $[a, b]$

1. Invoke $T_{\mathrm{BRC}}(a, b, k)$ and receive the output τ
2. **if** τ consists only of pair (y, d) **then**
3. $\quad \tau \leftarrow \langle (G_0(y), d-1), (G_1(y), d-1) \rangle$
4. Let $\tau = \langle (y_1, d_1), \ldots, (y_n, d_m) \rangle$ and $D = ((d_1, \ldots, d_c), (d_{c+1}, \ldots, d_m))$ be the corredponding decomposition of $r = b - a + 1$
5. **while** there is a maximum integer x in $\{0, \ldots, M_D\}$ that does not appear in D:
6. $\quad$ Find the rightmost pair $(y_i, x+1)$ and compute values $y_i^0 = G_0(y_i)$, $y_i^1 = G_1(y_i)$
7. $\quad$ Remove $(y_i, x+1)$ from τ, insert the pairs (y_i^0, x) and (y_i^1, x) in τ respecting the strictly decreasing order and update D accordingly
8. **if** the leftmost sequence of D is not $(B(r), \ldots, 0)$ **then**
9. $\quad$ Fill leftmost sequence with values from rightmost sequence until it complies with uniform decomposition of r
10. **return** τ

The process starts with invoking the T algorithm of BRC to get an initial trapdoor τ (Line 1). If the input range is covered by a single subtree, then τ is reformed into two pairs that correspond to the child subtrees,

so that the dephs in τ always form a decomposition (Lines 2,3). Let D be the decomposition implied by τ (Line 4). The loop in Lines 5-7 utilizes Theorem 2 and works as follows. It finds the highest depth x that does not exist in D, and "splits" the partial PRF value y_i in the rightmost pair in τ with depth $x+1$, producing two new partial PRF values with depth x, i.e., $y_i^0 = G_0(y_i)$ and $y_i^1 = G_1(y_i)$. (Line 4). Note that these values correspond to the two children of the subtree of y_i. Thus, if we substitute element $(y_i, x+1)$ by $(y_i^0, x), (y_i^1, x)$ in τ, then the trapdoor can still produce the same leaf PRF values as before. However, at all times we wish the sequence of depths in τ to form a decomposition. Hence, after removing $(y_i, x+1)$, we appropriately insert (y_i^0, x) in the left pair sequence and (y_i^1, x) in the right pair sequence of τ (Line 7). Upon termination of the loop, all the values in $\{0, \ldots, M_D\}$ appear in D and, thus, we have reached a w.c.d. according to Theorem 2. The process concludes, after properly re-arranging the elements of τ, such that they comply with the unique uniform decomposition of r (Lines 8,9). This is done deterministically, by simply filling the missing depths from $\{0, \ldots, B(r)\}$ in the left sequence with the unique appropriate pair that exists (by Theorem 2) in the right sequence.

In our running example, for the range $[2, 7]$, the T algorithm in URC converts the original token retrieved by the trapdoor algorithm of BRC, $\tau = \langle (f_k(001), 1), (f_k(01), 2) \rangle$, as follows (we underline a newly inserted element to the left sequence, and depict as bold a newly inserted element to the right sequence):

$$\langle (f_k(001), 1), (f_k(01), 2) \rangle$$
$$\downarrow$$
$$\langle \underline{(f_k(0010), 0)}, (f_k(01), 2), (\mathbf{f_k(0011)}, \mathbf{0}) \rangle$$
$$\downarrow$$
$$\langle \underline{(f_k(010), 1)}, (f_k(0010), 0), (\mathbf{f_k(011)}, \mathbf{1}), (f_k(0011), 0) \rangle$$

The C algorithm of URC is identical to BRC, because the trapdoor in URC has exactly the format expected by this algorithm, i.e., pairs of PRF values corresponding to GGM subtrees along with their depths. Moreover, during the evolution of the initial BRC trapdoor into one of uniform decomposition in the T algorithm of URC, a partial PRF value y is substituted by two new PRF values that can generate the same leaf PRF values as y. As such, the correctness of the BRC scheme is inherited in URC. Finally, due to Lemma 1, the size of a w.c.d. (and, hence, also a uniform decomposition) of r is $O(\log(r))$, which means that the trapdoor size in URC is also $O(\log(r))$.

4.3 Security

In this section we prove the security of BRC and URC. Both proofs rely on the security of the pseudorandom generator of the underlying GGM construction. Note that the security proof does not follow straightforwardly from the GGM proof because contrary to the case of GGM where the adversary obtains only *leaf* PRFs, the adversary in a DPRF can obtain also *partial* PRF values in the GGM tree (via trapdoor queries). Note that the tree structure of a trapdoor (which is independent of k) for a range predicate P of size r is deterministic and public in both BRC and URC. Thus, when querying the oracle in the security game, the adversary can map the partial or full PRF values appearing in τ for a selected P to subtrees in the GGM tree. Based on this observation, we prove first the security of BRC against adversaries that query only *subtrees*, or equivalently *prefixes* for strings in $\{0,1\}^n$, where n is the depth of the GGM tree. We call this type of security *subtree security*. We conclude our proofs by showing that the subtree security of BRC implies the security of both schemes.

In order to prove the subtree security property of BRC, we insure and exploit the subtree security for two special cases. First, we consider the case that the adversary is non-adaptive, i.e., it poses all its queries in advance.

Lemma 2 *The BRC scheme with depth n and maximum range size λ^γ is subtree secure against PPT adversaries that make all their queries, even the challenge query, non-adaptively.*

Proof. Let $\mathcal{A}$ be a non-adaptive prefix-only PPT adversary against a BRC scheme with depth n and maximum range size λ^γ. Without loss of generality we assume that $\mathcal{A}$ always advances to step 3 (submits a challenge to the challenger).

We define recursively two sequences of PPT algorithms $\mathcal{A} = \mathcal{A}_n, \ldots, \mathcal{A}_1$ and $S_n, \ldots, S_1$ as follows.

For $i = n-1, \ldots, 1$, $\mathcal{A}_i$ on input 1^λ, initially invokes $\mathcal{A}_{i+1}$ receiving all of its non-adaptive queries, and chooses a random value k' in $\{0,1\}^\lambda$. If a query $x_i \cdots x_t$ has the same most significant bit (MSB) as the challenge query, $\mathcal{A}_i$ makes the query $x_{i-1} \cdots x_t$, and responds with the received value y. Otherwise, it responds with the value $G_{x_t}(\cdots(G_{x_{i-1}}(k')))$. It returns $\mathcal{A}_{i+1}$'s output.

For $i = n, \ldots, 1$, on input $(z_0, z_1) \in \{0,1\}^{2\lambda}$, S_i invokes $\mathcal{A}_i$ and receives all of its queries. For every query $x_{i-1} \cdots x_t$, it responds with $G_{x_t}(\cdots(G_{x_{i-2}}(z_{x_{i-1}})))$ (or z_{x_0} for $i = 1$). On the challenge phase, it flips a coin b and acts as the challenger in a DPRF security game with $\mathcal{A}_i$. It returns 1 iff $\mathcal{A}_i$ returns b.

We denote by q_i and p_i the probability that S_i outputs 1 when it receives its input from the uniform distribution $U_{2\lambda}$ in $\{0,1\}^{2\lambda}$ and the pseudorandom distribution $G(U_\lambda)$, respectively. By the definition of S_i, $p_n = \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda) = 1]$ while $q_1 \leq 1/2$, since it corresponds to a totally random game.

We observe that $\mathcal{A}_{n-1}, \ldots, \mathcal{A}_1$ behave like attackers against the subtree security of BRC schemes with respective depths $n-1, \ldots, 1$. The behavior of A_i as an attacker is the same as A_{i+1}'s, when the latter interacts with a modified challenger that replaces the two possible partial PRF values for the MSB of a prefix, with two random values. Thus, following the previous notation, we have that $p_i = q_{i+1}$. Since $G(U_\lambda)$ and $U_{2\lambda}$ are indistinguishable, it holds that $|p_i - q_i| \leq \epsilon_i(\lambda)$, where $\epsilon_i(\cdot)$ is a negligible function. We also have

$$|p_n - q_1| = |\sum_{i=1}^{n}(p_i - q_i)| \leq \sum_{i=1}^{n}|p_i - q_i| \leq \sum_{i=1}^{n}\epsilon_i(\lambda),$$

hence $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda) = 1] = p_n \leq q_1 + n \cdot \epsilon(\lambda) \leq 1/2 + n \cdot \epsilon(\lambda)$, where $\epsilon(\lambda) = \max_i\{\epsilon_i(\lambda)\}$.

We use the above lemma to prove the security of a special category of BRC schemes, where the maximum range size is at least half of $A = [0, 2^n - 1]$, which represents the biggest interval where range predicates are defined. This will serve as a stepping stone for designing our final security proof.

Lemma 3 *The BRC scheme with depth n and maximum range size λ^γ is subtree secure if $2^{n-1} \leq \lambda^\gamma < 2^n$.*

Proof. Let $\mathcal{A}$ be a prefix-only adversary. We construct a non-adaptive prefix-only adversary $\mathcal{B}$ for the subtree security game that on input 1^λ chooses randomly a challenge x^* in $[0, 2^n - 1]$ and makes the n queries that cover all the possible values except from x^*. Namely, $\mathcal{B}$ makes queries $(x^*_{n-1} \oplus 1), \ldots, (x^*_{n-1} \cdots x^*_i \oplus 1), \ldots, (x^*_{n-1} \cdots x^*_0 \oplus 1)$ and submits challenge x^*. It receives responses $y_{n-1}, \ldots, y_0$ respectively, along with y^* which is the response to x^*. Then, it invokes $\mathcal{A}$ and plays the security game with $\mathcal{A}$, as a challenger that can respond appropriately for every value that is not x^* or a range that does not contain x^*. If $\mathcal{A}$ sets x^* as a challenge, then $\mathcal{B}$ responds with y^*, and returns $\mathcal{A}$'s guess. Otherwise, $\mathcal{A}$ has either made a query which is a prefix of x^*, or it has submitted a challenge different than x^*, so $\mathcal{B}$ terminates the game it plays with $\mathcal{A}$ and returns a random bit, as its guess to its challenge.

Let E be the event that $\mathcal{B}$ guesses $\mathcal{A}$'s challenge, i.e. $\mathcal{A}$'s challenge is x^*. By the description of $\mathcal{B}$ we have that

$$\Pr[\mathcal{G}^{\mathcal{B}}_{\mathsf{SEC}}(1^\lambda) = 1 \wedge \neg E] = 1/2 \cdot (1 - 1/2^n) \quad \text{and}$$
$$\Pr[\mathcal{G}^{\mathcal{B}}_{\mathsf{SEC}}(1^\lambda) = 1 \wedge E] = 1/2^n \cdot \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda) = 1].$$

Since $\mathcal{B}$ is non-adaptive, by Lemma 2 we get that for some negligible function $\epsilon(\cdot)$, $\Pr[\mathcal{G}^{\mathcal{B}}_{\mathsf{SEC}}(1^\lambda) = 1] \leq 1/2 + \epsilon(\lambda)$. By adding the above equations we have that

$$1/2^n \cdot (\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda) = 1] - 1/2) + 1/2 \leq 1/2 + \epsilon(\lambda),$$

so, $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda) = 1] \leq 1/2 + 2^n \cdot \epsilon(\lambda) \leq 1/2 + 2\lambda^\gamma \cdot \epsilon(\lambda)$.

We now apply Lemma 3 to prove the subtree security of BRC.

Lemma 4 *The BRC scheme with depth n and maximum range size λ^γ is subtree secure.*

Proof. See Appendix A.

Finally, we prove that the subtree security of BRC implies the security of BRC and URC.

Theorem 3 *The BRC and URC schemes with depth n and maximum range size λ^γ are secure.*

Proof. Let $\mathcal{A}$ be a PPT adversary that wins the DPRF security game of either BRC or URC with non-negligible advantage $\alpha(\cdot)$. As noted in the beginning of this section, any query $\mathcal{A}$ makes, can be represented as a sequence of $O(\log(\lambda^\gamma))$ subtrees, or equivalently of $O(\log(\lambda^\gamma))$ prefixes. Thus, we can construct a prefix-only adversary $\mathcal{A}'$ that invokes $\mathcal{A}$ and when it receives a query sequence $\langle x^1_{n-1} \cdots x^1_{t^1}, \ldots, x^m_{n-1} \cdots x^m_{t^m} \rangle$ from $\mathcal{A}$, it makes all prefix queries separately, receives $y_1, \ldots, y_m$ and answers by $\langle y_1, \ldots, y_m \rangle$. $\mathcal{A}'$ also transfers $\mathcal{A}$'s challenge and outputs its guess. Therefore, it wins the DPRF security game with advantage $\alpha(\cdot)$, which contradicts Lemma 4.

4.4 Policy Privacy

This section analyzes the policy privacy of URC. According to the lower bound argument we gave in Section 3, URC cannot be expected to satisfy the general policy privacy property, because it is efficient. We illustrate this explicitly with a toy example. For challenge ranges $[2, 5]$ and $[4, 7]$, the trapdoors will contain PRF values corresponding to subtrees covering the ranges as $[2, 3], \{4\}, \{5\}$ and $[4, 5], \{6\}, \{7\}$, respectively. Therefore, the adversary can issue query for leaf 4 and receive a PRF value y. Having $\langle (y_1, 1), (y_2, 0), (y_3, 0) \rangle$ as challenge trapdoor, it can check whether $y_2 = y$, which happens only when $[2, 5]$ was chosen by the challenger.

Nevertheless, in the theorem below we prove that URC achieves *union policy privacy*. The above attack is circumvented as in the union policy privacy game, the adversary cannot obtain a PRF value for a leaf in the *intersection* of the challenge ranges, i.e., for 4 and 5.

Theorem 4 *The URC scheme with depth n and maximum range size λ^γ is a DPRF with union policy privacy.*

Proof. See Appendix A.

5 Applications

In this section we discuss interesting applications of the general DPRF primitive and our specialized range constructions. We stress, though, that their applicability is not limited to these scenarios; we are confident that they can capture a much wider set of applications.

Authentication and access control in RFID. *Radio Frequency Identification* (RFID) is a popular technology that is expected to become ubiquitous in the near future. An RFID *tag* is a small chip with an antenna. It typically stores a unique ID along with other data, which can be transmitted to a *reading device* lying within a certain range from the tag. Suppose that a *trusted center* (TC) possesses a set of RFID tags (attached to books, clothes, etc), and distributes RFID readers to specified locations (e.g., libraries, campuses, restaurants, etc.). Whenever a person or object carrying a tag lies in proximity with a reader, it transmits its data (e.g., the title of a book, the brand of a jacket, etc.). The TC can then retrieve these data from the RFID readers, and mine useful information (e.g., *hotlist* books, clothes, etc.).

Despite its merits, RFID technology is challenged by security and privacy issues. For example, due to the availability and low cost of the RFID tags, one can easily create tags with arbitrary information. As such, an adversary may *impersonate* other tags, and provide falsified data to legitimate readers. On the other hand, a reader can receive data from any tag in its vicinity. Therefore, sensitive information may be leaked to a reader controlled by an adversary. For example, the adversary may learn the ID and the title of a book stored in a tag, match it with public library records, and discover the identity and reading habits of an individual.

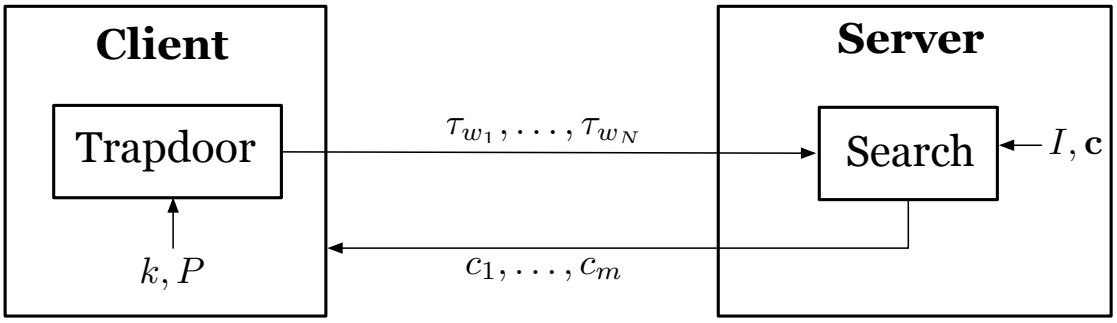
Motivated by the above, the literature has addressed *authentication* and *access control* in RFID. A notable paradigm was introduced in [30], which can be directly benefited by DPRFs. At a high level, every tag is associated with a key, and the TC *delegates* to a reader a set of these keys (i.e., the reader is authorized to authenticate and access data from only a subset of the tags). The goal is for the TC to reduce certain costs, e.g., the size of the delegation information required to *derive* the tag keys while maintaining a high number of distinct keys in order to ensure that attacks can be compartmentalized.

Observe that a DPRF $(\mathcal{F}, T, C)$ is directly applicable to the above setting. $\mathcal{F}$ is defined on the domain of the tag IDs, and its range is the tag keys. Given a delegation predicate on the tag IDs, the TC generates a trapdoor via algorithm T, and sends it to the reader. The latter runs C on the trapdoor to retrieve the tag keys. In fact, for the special case where the access policy is a *range* of IDs, the delegation protocol suggested in [30] is identical to the non-private BRC scheme (we should stress though that [30] lacks rigorous definitions and proofs). Range policies are meaningful, since tag IDs may be sequenced according to some common theme (e.g., books on the same topic are assigned consecutive tag IDs). In this case, a range policy *concisely* describes a set of tags (e.g., books about a certain religion) and, hence, the system can enjoy the logarithmic delegation size of BRC. However, as explained in Section 4, BRC leaks the position of the IDs in the tree, which may further leak information about the tags. Although [30] addresses tag privacy, it provides no privacy formulation, and overlooks the above structural leakage. This can be mitigated by directly applying our policy-private URC construction for delegating tag keys to the readers. To sum up, DPRFs find excellent application in authentication and access control in RFIDs, enabling communication-efficient tag key delegation from the TC to the reader. Moreover, policy-private DPRFs provide a higher level of protection for the tag IDs against the readers.

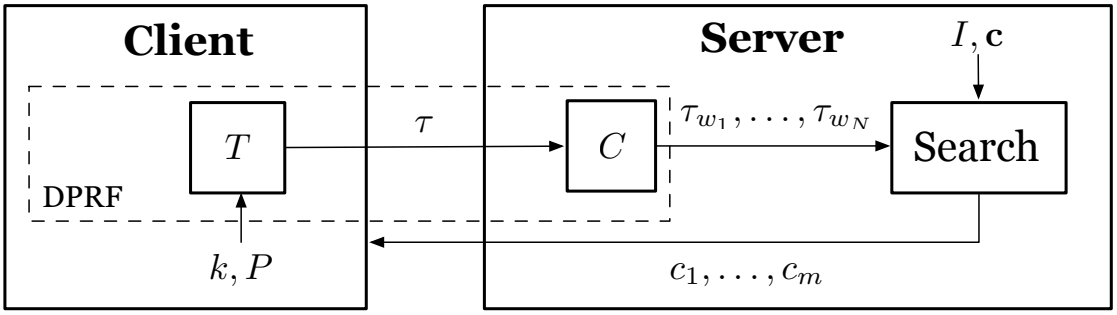
Batch queries in symmetric searchable encryption. Symmetric searchable encryption (SSE) [11, 24] enables queries to be processed directly on ciphertexts generated with symmetric encryption. Although SSE is a general paradigm, here we focus on the definitions and schemes of [11]. Previous works support the special case of *keyword* queries, and provide an acceptable level of provable security. The general framework underlying [11] is as follows. In an offline stage, a client encrypts his data with his secret key k, and uploads the ciphertexts $\mathbf{c}$ to an untrusted server. He also creates and sends a *secure index* I on the data for efficient keyword search, which is essentially an encrypted lookup table or inverted index. Given a keyword w, the client generates a query *token* τ_w using k, and forwards it to the server. It is important to stress that this "trapdoor" is merely comprised of one or more PRF values computed on w with k, which were used as keys to encrypt I. For simplicity and w.l.o.g., we assume that τ_w is a single PRF value. The server uses

τ_w on I and retrieves the IDs of the ciphertexts associated with w. The results $c_1, \ldots, c_m$ are retrieved and transmitted back to the client, who eventually decrypts them with his key. The security goal is to protect *both* the data and the keyword from the server.

Suppose that the client wishes to search for a *batch* of N keywords $w_1, \ldots, w_N$. For instance, the client may ask for documents that contain *multiple* keywords of his choice (instead of just a single one). As another example, assume that the client's data are employee records, and each record contains a salary attribute that takes as values intervals of the form [iK, $(i+1)$K] (i.e., instead of exact salaries). These intervals can serve as keywords in SSE search. Suppose that the client wishes to retrieve records with salaries in a *range* of intervals, e.g., [1K, 10K]. Observe that this cannot be processed with a single keyword query (no record is associated with [1K, 10K]). To overcome this while utilizing the SSE functionality, the client can ask 9 *distinct* queries with keywords "[1K, 2K]", "[2K, 3K]", . . ., "[9K, 10K]", which cover the query range [1K, 10K]. Such scenarios are handled with "traditional" SSE as shown in Figure 3(a). Given a predicate P that describes the keywords $w_1, \ldots, w_N$ in the batch query, the client generates N trapdoors $\tau_{w_1}, \ldots, \tau_{w_N}$ using the standard SSE trapdoor algorithm, and sends them to the server. The server then searches I with every τ_{w_i} following the SSE protocol. Apparently, for large N, the computational and communication cost at the client is greatly impacted.



(a) "Traditional" SSE



(b) SSE augmented with DPRFs

Figure 3: Batch keyword query processing in SSE

We can augment the discussed SSE framework with DPRF functionality, in order to support batch queries with *sublinear* (in N) processing and communication cost at the client, while providing provable security along the lines of [24]. Figure 3(b) illustrates our enhanced SSE architecture. Recall that τ_{w_i}

is practically a PRF on w_i produced with key k. Therefore, instead of computing a PRF value for every w_i himself, the client *delegates* the computation of these PRF values to the server by employing a DPRF $(\mathcal{F}, T, C)$, where $\mathcal{F}$ is defined over the domain of the keywords. Given predicate P and k, the client runs T and generates trapdoor τ, which is sent to the server. The latter executes C on τ to produce $\tau_{w_1}, \ldots, \tau_{w_N}$. Execution then proceeds in the same manner as in "traditional" SSE. Observe that, for the special case of ranges, if URC is used as the DPRF, the computational and communication cost at the client decreases from $O(N)$ to $O(\log(N))$. This transformation would work "out of the box" in combination to any SSE scheme that uses a PRF for creating tokens.

The above framework can be proven secure against *adaptive* adversaries along the lines of [11]. The most important alteration in the security game and proof is the formulation of the *information leakage* of the C algorithm of the DPRF. In particular the level of keyword privacy that is provided by the construction is dictated by the level of policy-privacy that is attained by the underlying DPRF. Specifically, given a DPRF with multi-instance policy-privacy, it is easy to show that the same level of keyword privacy as in [11] can be attained. Weaker notions of policy privacy (such as single-instance or union) result in correspondingly weaker notions of keyword privacy. For instance, recall that union policy privacy ensures indistinguishability of queries not intersecting with previously asked queries. Thus, combining PRF-based SSE with our URC construction will provide this level of keyword privacy. Given the lower bound arguments for tree-wise constructions we sketched in Section 3, the privacy loss incurred by this tree-based design is the unavoidable cost for the exponential efficiency improvement in client to server communication. Efficient constructions attaining higher level of policy privacy might be feasible but they will have to follow a non tree-base design paradigm.

Broadcast encryption. In a broadcast encryption scheme, cf. [14, 31, 32] a sender wishes to transmit data to a set of receivers so that at each transmission the set of receivers excluded from the recipient set can be chosen on the fly by the sender. In particular this means that the sender has to be able to make an initial key assignment to the recipients and then suitably use the key material so that only the specific set of users of its choosing can receive the message. In such schemes it was early on observed that there is a natural tradeoff between receiver memory storage and ciphertext length (e.g., see lower bounds in [26]). The intuition behind this is that if the receivers have more keys this gives to the sender more flexibility in the way it can encrypt a message to be transmitted.

In the above sense one can think of the key assignment step of a broadcast encryption scheme as a PRF defined over the set Φ which contains all distinct subsets that the broadcast encryption scheme assigns a distinct key. Given this configuration the user u will have to obtain all the keys corresponding to subsets $S \in \Phi$ for which it holds that $u \in S$ (we denote those subsets by $S_u \subseteq \Phi$). In DPRF language this would correspond to a delegation policy for a PRF: users will need to store the trapdoor that enables the evaluation of any value in the delegated key set S_u.

Seen in this way, *any* DPRF is a key assignment mechanism for a broadcast encryption scheme that saves space on receiver storage. For example our construction for ranges gives rise to the following broadcast encryption scheme: receivers $[n] = \{1, \ldots, n\}$ are placed in sequence; each receiver $u \in [n]$ is initialized with the trapdoor for a range $[u - t, u + t]$ for some predetermined parameter $t \in \mathbb{Z}$. In this broadcast encryption scheme the sender can very efficiently enable any range of receivers that is positioned at distance at most t from a fixed location $v \in [n]$. This is done with a single ciphertext (encrypted with the key of location v). Any receiver of sufficient proximity t to location v can derive the corresponding decryption key from its trapdoor. Furthermore, given the efficiency of our construction, storage on receivers is only logarithmic on t. While the semantics of this scheme are more restricted than a full-fledged broadcast encryption scheme (which enables the sender to exclude any subset of users on demand) it succinctly illustrates the relation be-

tween broadcast encryption and DPRF; further investigation in the relation between the two primitives from a construction point of view will be motivated by our notion. Specifically, an efficient DPRF with domain Φ over a policy set $\mathcal{P} = \{S_u \mid u \in [n]\}$ will provide an efficient key assignment for a broadcast encryption scheme operating over Φ. Interestingly, the reverse is also true and a broadcast encryption scheme with efficient key assignment will provide a DPRF with domain Φ and the policy set $\{S_u \mid u \in [n]\}$.

Regarding policy privacy, it is interesting to point out that this security property is yet unstudied in the domain of broadcast encryption. A different privacy notion [3, 13, 25] was considered that deals with the structure of ciphertexts in such schemes. Our policy privacy on the other hand deals with the privacy of memory contents from the receiver point of view. Maintaining the indistinguishability of the storage contents is a useful security measure in broadcast encryption schemes and our DPRF primitive will motivate the study of this security property in the context of braodcast encryption (note that none of the tree-like key-delegation methods used in broadcast encryption schemes prior to our work satisfy policy privacy).

6 Conclusion

We have introduced the concept of *delegatable pseudorandom functions* (DPRFs), a new cryptographic primitive that allows for policy-based computation at an untrusted proxy of PRF values without knowledge of a secret or even the input values. We provided formal definitions of the core properties of DPRFs for (1) correctness, the ability of the proxy to compute PRF values only for inputs that satisfy a given predicate, (2) security, the standard pseudorandomness guarantee but against a stronger adversary that also issues delegation queries, and (3) policy privacy, preventing leakage of the secret preimages of the computed PRF values. Moreover, we presented two DPRF constructions along with a comprehensive analysis in terms of their security and privacy guarantees and some inherent trade-offs with efficiency. Our proposed DPRFs are generic, yet practical, based on the well-understood and widely-adopted GGM design framework for PRFs and, as we showed, they find direct application in many key delegation or derivation settings providing interesting new results. Further exploration of DPRFs holds promise for new interesting results. Open problems include: Designing DPRFs for other classes of predicates, establishing upper and lower bounds on the connection between efficiency and policy privacy, and studying applications in other settings.

References

[1] M. J. Atallah and K. B. Frikken. Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 48–59, New York, NY, USA, 2010. ACM.

[2] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, Feb. 2006.

[3] A. Barth, D. Boneh, and B. Waters. Privacy in encrypted content distribution using private broadcast encryption. In *Financial Cryptography*, pages 52–64, 2006.

[4] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *Proceedings of the 31st annual conference on Advances in cryptology*, CRYPTO'11, pages 111–131, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, 1998.

[6] A. Boldyreva, A. Palacio, and B. Warinschi. Secure proxy signature schemes for delegation of signing rights. *Journal of Cryptology*, 25:57–115, 2012. 10.1007/s00145-010-9082-x.

[7] D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. *IACR Cryptology ePrint Archive*, 2013:352, 2013.

[8] E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013.

[9] R. Canetti, B. Riva, and G. N. Rothblum. Practical delegation of computation using multiple servers. In *ACM Conference on Computer and Communications Security*, pages 445–454, 2011.

[10] K.-M. Chung, Y. T. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[11] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *CCS*, 2006.

[12] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In S. Vaudenay, editor, *Public Key Cryptography - PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer Berlin / Heidelberg, 2005.

[13] N. Fazio and I. M. Perera. Outsider-anonymous broadcast encryption with sublinear ciphertexts. In *Public Key Cryptography*, Lecture Notes in Computer Science. Springer, 2012.

[14] A. Fiat and M. Naor. Broadcast encryption. In *CRYPTO*, pages 480–491, 1993.

[15] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *ACM Conference on Computer and Communications Security*, pages 501–512, 2012.

[16] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.

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

[18] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[19] M. Green and G. Ateniese. Identity-based proxy re-encryption. *IACR Cryptology ePrint Archive*, 2006:473, 2006.

[20] S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 264–282, Berlin, Heidelberg, 2005. Springer-Verlag.

[21] G. Itkis. *Handbook of Information Security*, chapter Forward Security: Adaptive Cryptography—Time Evolution. John Wiley and Sons, 2006.

[22] A.-A. Ivan and Y. Dodis. Proxy cryptography revisited. In *NDSS*, 2003.

[23] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 577–594, Berlin, Heidelberg, 2009. Springer-Verlag.

[24] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *CCS*, 2012.

[25] B. Libert, K. G. Paterson, and E. A. Quaglia. Anonymous broadcast encryption: Adaptive security and efficient constructions in the standard model. In *Public Key Cryptography*, Lecture Notes in Computer Science. Springer, 2012.

[26] M. Luby and J. Staddon. Combinatorial bounds for broadcast encryption. In *EUROCRYPT*, pages 512–526, 1998.

[27] A. Lysyanskaya. Unique signatures and verifiable random functions from the dh-ddh separation. In *CRYPTO*, pages 597–612, 2002.

[28] M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures for delegating signing operation. In *Proceedings of the 3rd ACM conference on Computer and communications security*, CCS '96, pages 48–57, New York, NY, USA, 1996. ACM.

[29] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.

[30] D. Molnar, A. Soppera, and D. Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of rfid tags. In *Proceedings of the 12th international conference on Selected Areas in Cryptography*, SAC'05, pages 276–290, Berlin, Heidelberg, 2006. Springer-Verlag.

[31] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, pages 41–62, 2001.

[32] M. Naor and B. Pinkas. Efficient trace and revoke schemes. *Int. J. Inf. Sec.*, 9(6):411–424, 2010.

[33] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: verifiable computation from attribute-based encryption. In *Proceedings of the 9th international conference on Theory of Cryptography*, TCC'12, pages 422–439, Berlin, Heidelberg, 2012. Springer-Verlag.

Proof of Lemma 4

By Lemma 3, it suffices to show for the case that $\lambda^\gamma < 2^{n-1}$. This is definitely the interesting case, since 2^n is normally superpolynomial in λ. Let $\mathcal{A}$ be a prefix-only adversary against BRC that makes at most $q(\lambda)$ queries (including the challenge) and d be the minimum integer that $\lambda^\gamma < 2^d$. We construct a PPT PRF distinguisher $\mathcal{B}$ that makes oracle queries of fixed size $n' = n - d \geq 1$. On input 1^λ, $\mathcal{B}$ flips a coin b, invokes $\mathcal{A}$ and initializes a security game, itself being the challenger. By the bound on the size of the ranges, all queries of $\mathcal{A}$ have length greater than n'.Thus, for every query $x_{n-1} \cdots x_t$, we have that $t < d$ and $\mathcal{B}$ responds by making a query $x_{n-1} \cdots x_d$, which is of length n', receiving a value y and answering to $\mathcal{A}$ as $G_{x_t}(\cdots(G_{x_{d-1}}(y)))$. When $\mathcal{A}$ submits a challenge, $\mathcal{B}$ acts as a normal challenger according to the coin flip b utilizing its oracle to determine the value up to level d as above. Finally, $\mathcal{B}$ returns 1 iff $\mathcal{A}$ returns b. Clearly,

when $\mathcal{B}$'s oracle is a PRF f_k of length n', $\mathcal{B}$ returns 1 iff $\mathcal{A}$ wins, i.e $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda) = 1] = \Pr[\mathcal{B}^{f_k(\cdot)}(1^\lambda) = 1]$. Since f_k is a PRF, we have that for some negligible function $\epsilon(\cdot)$,

$$|\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda) = 1] - \Pr[\mathcal{B}^{R(\cdot)}(1^\lambda) = 1]| \leq \epsilon(\lambda). \tag{1}$$

Consequently, we construct a prefix-only adversary $\tilde{\mathcal{A}}$ against a BRC with depth d and maximum range size λ^γ as follows: $\tilde{\mathcal{A}}$ invokes $\mathcal{A}$ and chooses a random index $j \in [q(\lambda)]$ and $q(\lambda) - 1$ random values $k_1, \ldots, k_{j-1}, k_{j+1}, \ldots, k_{q(\lambda)} \in \{0,1\}^\lambda$. Index j reflects $\tilde{\mathcal{A}}$'s attempt to guess which of all of possible different prefixes of length n' that will appear in $\mathcal{A}$'s queries will be the one that a prospective challenge query will have. Then $\tilde{\mathcal{A}}$ keeps count of the different prefixes of length n' that gradually appear and reacts to a query $x_{n-1} \cdots x_t$ according to the following checks:

(i). $x_{n-1} \cdots x_d$ is the i-th prefix and $i \neq j$: $\tilde{A}$ responds with $G_{x_t}(\cdots(G_{x_{d-1}}(k_i)))$.

(ii). $x_{n-1} \cdots x_d$ is the j-th prefix: if all of the queries made by $\mathcal{A}$ that have the j-th prefix, along with $x_{n-1} \cdots x_t$, cover the whole subtree with root prefix $x_{n-1} \cdots x_d$, T_j, then $\tilde{A}$ terminates the game with $\mathcal{A}$ and chooses a leaf $z_{d-1} \cdots z_0$ of T_j that has not been covered by $\mathcal{A}$'s previous queries. It submits $z_{d-1} \cdots z_0$ as its challenge and returns a random bit. Otherwise, $\tilde{A}$ makes query $x_{d-1} \cdots x_t$ and responds with the received value y.

(iii). $t = 0$ and $x_{n-1} \cdots x_0$ is $\mathcal{A}$'s challenge: if $x_{n-1} \cdots x_d$ is not the j-th prefix, then $\tilde{\mathcal{A}}$ terminates the game with $\mathcal{A}$, chooses a leaf $z_{d-1} \cdots z_0$ of T_j not yet covered by $\mathcal{A}$'s queries, submits $z_{d-1} \cdots z_0$ as its challenge and returns a random bit. Otherwise, it submits challenge $x_{d-1} \cdots x_0$, receives value y^* and responds with y^*.

If $\tilde{\mathcal{A}}$ does not terminate the security game with $\mathcal{A}$, then it returns $\mathcal{A}$'s guess. By the choice of d, $2^{d-1} \leq \lambda^\gamma < 2^d$, hence Lemma 3 implies that $\tilde{\mathcal{A}}$ has negligible distinguishing advantage. Thus for some negligible function $\delta(\cdot)$,

$$\Pr[\mathcal{G}^{\tilde{\mathcal{A}}}_{\mathsf{SEC}}(1^\lambda) = 1] \leq 1/2 + \delta(\lambda). \tag{2}$$

By the description of $\tilde{\mathcal{A}}$, the interaction between $\mathcal{A}$ and $\mathcal{B}$ in the case that $\mathcal{B}$'s oracle is random is fully simulated by $\tilde{\mathcal{A}}$ when the latter's guess for the prefix of $\mathcal{A}$'s challenge is correct. Formally, E be the event that $\tilde{\mathcal{A}}$ guesses $\mathcal{A}$'s challenge. Then it holds that

$$\Pr[\mathcal{B}^{R(\cdot)}(1^\lambda) = 1] = \Pr[\mathcal{G}^{\tilde{\mathcal{A}}}_{\mathsf{SEC}}(1^\lambda) = 1 | E]. \tag{3}$$

By the description of $\tilde{\mathcal{A}}$ and (3), we have that

$$\Pr[\mathcal{G}^{\tilde{\mathcal{A}}}_{\mathsf{SEC}}(1^\lambda) = 1 \wedge \neg E] = 1/2 \cdot (1 - 1/q(\lambda)) \quad \text{and}$$
$$\Pr[\mathcal{G}^{\tilde{\mathcal{A}}}_{\mathsf{SEC}}(1^\lambda) = 1 \wedge E] = 1/q(\lambda) \cdot \Pr[\mathcal{B}^{R(\cdot)}(1^\lambda) = 1],$$

where we used that $\Pr[E] = 1/q(\lambda)$. Therefore,

$$\Pr[\mathcal{G}^{\tilde{\mathcal{A}}}_{\mathsf{SEC}}(1^\lambda) = 1] = 1/2 + 1/q(\lambda) \cdot (\Pr[\mathcal{B}^{R(\cdot)}(1^\lambda) = 1] - 1/2),$$

so by applying (2) we get

$$\Pr[\mathcal{B}^{R(\cdot)}(1^\lambda) = 1] \leq 1/2 + q(\lambda) \cdot \delta(\lambda). \tag{4}$$

Finally, by (1) and (4) we conclude that

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{SEC}}(1^\lambda) = 1] \leq 1/2 + q(\lambda) \cdot \delta(\lambda) + \epsilon(\lambda) =$$
$$= 1/2 + \mathrm{negl}(\lambda).$$

Proof of Theorem 4

Let $\mathcal{A}$ be a PPT adversary that wins the union policy privacy game with non-negligible advantage $\alpha(\cdot)$. Let P_0, P_1 be the two challenge ranges and b the chosen random bit, hence $\mathcal{A}$ receives the challenge trapdoor τ_b for P_b. Denote each element of a decomposition D as (x, L) or (x, R), if it is integer x and belongs to the leftmost or rightmost sequence of D respectively. Define the ordering $<_D$ over the elements of D as follows: $(x, \mathrm{L}) <_D (y, \mathrm{R})$ or $(x, \mathrm{R}) <_D (y, \mathrm{L})$, if $x < y$ and $(x, \mathrm{L}) <_D (x, \mathrm{R})$. We define a sequence of hybrid games $\mathcal{G}_0^{\mathcal{A}}(1^\lambda), \ldots, \mathcal{G}_N^{\mathcal{A}}(1^\lambda)$, where N is the maximum size of a trapdoor output by T_{URC}. As shown in section 4.2, $N = 2\lceil \log(\lambda^\gamma + 2) \rceil - 1$. The game $\mathcal{G}_i$ is executed as the original game $\mathcal{G}_{\mathsf{UPP}}^{\mathcal{A}}(1^\lambda)$ during the pre-challenge and the post-challenge phase. Assume the pairs of the challenge trapdoor τ_b are arranged according to the ordering determined by the decomposition formed by the depths. The only modification in $\mathcal{G}_i^{\mathcal{A}}(1^\lambda)$ is that in the first i pairs of τ_b, the partial PRF values are the same as τ_b's, while all the other elements are replaced by random values in $\{0,1\}^\lambda$. In $\mathcal{G}_0^{\mathcal{A}}(1^\lambda)$, the challenge trapdoor consists of a number of pairs of random values attached to certain integers, independently of the choice of b. Therefore, $\Pr[\mathcal{G}_0^{\mathcal{A}}(1^\lambda) = 1] = 1/2$. Since $\mathcal{G}_N^{\mathcal{A}}(1^\lambda)$ is the UPP game, it holds that

$$\Pr[\mathcal{G}_N^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathcal{G}_0^{\mathcal{A}}(1^\lambda) = 1] \geq \alpha(\lambda).$$

Let E_r be the event that the size of the challenge ranges $|A_{P_0}|$, $|A_{P_1}|$ that $\mathcal{A}$ submits is r. Then for some challenge bit $b \in \{0,1\}$ and $r \in [\lambda^\gamma]$:

$$\Pr[\mathcal{G}_N^{\mathcal{A}}(1^\lambda) = 1 \wedge b \wedge E_r] - \Pr[\mathcal{G}_0^{\mathcal{A}}(1^\lambda) = 1 \wedge b \wedge E_r] \geq \alpha(\lambda)/2\lambda^\gamma,$$

which implies that there exists an $i \in [\lambda^\gamma]$ such that

$$\begin{aligned} &\Pr[\mathcal{G}_i^{\mathcal{A}}(1^\lambda) = 1 \wedge b \wedge E_r] - \\ &\quad - \Pr[\mathcal{G}_{i-1}^{\mathcal{A}}(1^\lambda) = 1 \wedge b \wedge E_r] \geq \alpha(\lambda)/2N\lambda^\gamma. \end{aligned} \tag{5}$$

We will show that for these fixed b, r, i, we can construct an adversary $\mathcal{A}_i$ for the security game of a BRC construction with depth that has non-negligible winning advantage. The main idea is that $\mathcal{A}_i$ invokes $\mathcal{A}$ and simulates either $\mathcal{G}_i$ or $\mathcal{G}_{i-1}$ on selected challenge P_b of size r depending on the value of the challenge bit b_i for the security game $\mathcal{G}_{\mathsf{SEC}}^{\mathcal{A}_i}(1^\lambda)$.

On input 1^λ, $\mathcal{A}_i$ computes the uniform decomposition of r, U_r, and arranges its elements according to $<_{U_r}$. Let u_i be the integer that appears in the i-th element of U_r. The BRC that $\mathcal{A}_i$ attacks has depth $n - u_i$. Specifically, $\mathcal{A}_i$ invokes $\mathcal{A}$ and answers all of its pre-challenge queries as follows: for each query $x_{n-1} \cdots x_t$, if $t \geq u_i$, it just transfers the query, receives value y, and responds with y. Otherwise, it makes query $x_{n-1} \cdots x_{u_i}$, receives value y, and responds with $G_{x_t}(\cdots(G_{x_{u_i-1}}(y)))$. In the challenge phase, if $|A_{P_b}| \neq r$, then $\mathcal{A}_i$ terminates the game with $\mathcal{A}$, chooses a valid random challenge, and returns a random bit. Otherwise, it makes $i-1$ queries and computes the $<_{U_r}$-first $i-1$ partial delegation keys $y_1, \ldots, y_{i-1}$ of τ_b as in the pre-challenge phase, and sets the string $x_{n-1}^* \cdots x_{u_i}^*$ that corresponds to the i-th partial key as its challenge, receiving y^*. Note that since $\mathcal{A}$ is restricted from making queries within $A_{P_0} \cup A_{P_1}$, it makes no queries with prefix $x_{n-1}^* \cdots x_{u_i}^*$, thus $\mathcal{A}_i$'s challenge is valid. It arranges the values $y_1, \ldots, y_{i-1}, y^*$ according to the order that τ_b imposes and "fills" the $|U_r| - i$ remaining positions of an array like trapdoor τ_b with $|U_r| - i$ pairs consisting of random values from $\{0,1\}^\lambda$ along with the corresponding depths. It returns τ_b to $\mathcal{A}$ and answers to $\mathcal{A}$'s post-challenge queries as in the pre-challenge phase. If $\mathcal{A}$ returns b, then $\mathcal{A}_i$ returns 1, otherwise it returns a random bit.

The probability that $\mathcal{A}_i$ wins the security game is

$$\begin{aligned}\Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda)=1] = \Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda)=1 \wedge \neg E_r] + \\ + \Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda)=1 \wedge E_r \wedge b_i = 1] + \\ + \Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda)=1 \wedge E_r \wedge b_i = 0].\end{aligned} \tag{6}$$

It holds that $\Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda)=1 \wedge \neg E_r] = 1/2 \cdot (1 - \Pr[E_r])$.

For the other two terms in the right part of (6), we observe that when $b_i = 1$, $\mathcal{A}_i$ simulates $\mathcal{G}_i$ when b and E_r occur, whereas when $b_i = 0$, $\mathcal{A}_i$ simulates $\mathcal{G}_{i-1}$ when b and E_r occur. Therefore, by the description of $\mathcal{A}_i$ we have that

$$\begin{aligned}\Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda)=1 \wedge E_r \wedge b_i = 1] = \\ = 1 \cdot \Pr[\mathcal{G}^{\mathcal{A}}_i(1^\lambda)=1 \wedge b \wedge E_r] + \\ + 1/2 \cdot \Pr[\mathcal{G}^{\mathcal{A}}_i(1^\lambda) \neq 1 \wedge b \wedge E_r] = \\ = 1/2 \cdot \Pr[b \wedge E_r] + 1/2 \cdot \Pr[\mathcal{G}^{\mathcal{A}}_i(1^\lambda)=1 \wedge b \wedge E_r]).\end{aligned}$$

and

$$\begin{aligned}\Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda)=1 \wedge E_r \wedge b_i = 0] = \\ = 0 \cdot \Pr[\mathcal{G}^{\mathcal{A}}_{i-1}(1^\lambda)=1 \wedge b \wedge E_r] + \\ + 1/2 \cdot \Pr[\mathcal{G}^{\mathcal{A}}_{i-1}(1^\lambda) \neq 1 \wedge b \wedge E_r] = \\ = 1/2 \cdot \Pr[b \wedge E_r] - 1/2 \cdot \Pr[\mathcal{G}^{\mathcal{A}}_{i-1}(1^\lambda)=1 \wedge b \wedge E_r].\end{aligned}$$

By the independency of b and E_r, we have that $\Pr[b \wedge E_r] = 1/2 \cdot \Pr[E_r]$. Thus, we evaluate $\Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda) = 1]$ according to (6) as

$$\begin{aligned}\Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda)=1] = 1/2 + 1/2 \cdot (\Pr[\mathcal{G}^{\mathcal{A}}_i(1^\lambda)=1 \wedge b \wedge E_r] - \\ - \Pr[\mathcal{G}^{\mathcal{A}}_{i-1}(1^\lambda)=1 \wedge b \wedge E_r]).\end{aligned}$$

Therefore by (5), $\Pr[\mathcal{G}^{\mathcal{A}_i}_{\mathsf{SEC}}(1^\lambda)=1] \geq 1/2 + \alpha(\lambda)/4N\lambda^\gamma$, which contradicts Theorem 3.