Delegatable Pseudorandom Functions and Applications

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

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

Stavros Papadopoulos
Dept. of Computer Science & Engineering
HKUST, Hong Kong
stavrosp@cse.ust.hk

Thomas Zacharias
Dept. of Informatics & Telecommunications
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 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Categories and Subject Descriptors

C.2.4 [Communication Networks]: Distributed Systems—client/server, distributed applications; E.3 [Data Encryption]; G.3 [Probability and Statistics]: random number generation; K.6.5 [Management of Computing and Information Systems]: Security and Protection—authentication, cryptographic controls

General Terms

Algorithms, Security, Theory

Keywords

delegation of computation; pseudorandom functions; RFIDs; searchable encryption; broadcast encryption; authentication

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).

Surprisingly enough, pseudorandom functions (PRFs), a fundamental primitive for emulating perfect randomness via keyed functions, thus, finding numerous applications in information security, have not been explicitly studied in the context of delegation of computations (of PRF values). We, hereby, initiate a study on this matter.

A new PRF concept. We introduce delegatable pseudorandom functions, or DPRFs for short, a novel cryptographic primitive 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
Delegator \( k, P \) to \( P \), which provides the elementary functionality shown in Figure 1: For any secret key \( k \) and a predicate \( P \in P \), the delegator computes a trapdoor \( \tau \) via algorithm \( T \), and \( \tau \) is transmitted to the proxy. The latter then runs algorithm \( C \) on \( \tau \) to derive exactly the set of PRF values \( B_P = \{ f_k(x) \mid P(x) \} \), where \( f_k \in F \), 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 \( A_P = \{ x \mid P(x) \} \).

![Diagram](image)

**Figure 1:** The DPRF functionality.

What motivates the above scheme is bandwidth efficiency: As long as the trapdoor \( \tau \) 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 \( \tau \).

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 (set \( A_P \)) specified by predicate \( P \). Security captures the requirement that the delegation capabilities of the PRF do not compromise its core pseudorandomness property. But this condition goes beyond the standard security definition of PRFs, since the pseudorandomness attacker, in addition to the PRF oracle, may now adaptively query also a trapdoor oracle for delegation on 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 \( B_P \)). Our security notion postulates that for any two predicates \( P, P' \) the corresponding trapdoors are indistinguishable, provided that \( |A_P| = |A_{P'}| \) and the adversary issues no delegation queries that trivially separate \( B_P \) and \( B_{P'} \).

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 \( j \)-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 \( \tau \) comprise a subset \( I_P \) of internal PRG values that 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 \( \tau \) 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 \( |B_P| \) (by which 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 RFIDds, 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 for future work. Selected proofs appear in the Appendix.

2. RELATED WORK

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.

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 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.

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

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/delegation queries and addresses policy-privacy issues.

3. DEFINITIONS

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

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

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

Delegatable PRFs. The notion of delegation for a PRF family \(\mathcal{F}\) as above is defined with respect to a delegation policy \(P\), i.e., \(P\) is a set of predicates defined over \(A\), also indexed by \(\lambda\), 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 \((F, T, C)\) is a delegatable PRF, or DPRF scheme, for family \(\mathcal{F} = \{f_k : A \rightarrow B \mid k \in \mathbb{K}\}\) with respect to policy \(P\) of predicates over \(A\), provided it satisfies two properties, correctness and security, that are defined individually below.

Correctness. We define \(T\) to be a PPT algorithm such that, given a description of \(P \in \mathcal{P}\) and a key \(k \in \mathbb{K}\), it outputs a “trapdoor” \(\tau\). The latter is intended to be used along with the deterministic PPT algorithm \(C\) for the computation of every element of \(A\) that satisfies the predicate \(P\). For fixed \(P, k\), the algorithm \(C\) can be viewed as a function

\[
C : S_{T,P,k} \rightarrow B \times S_{T,P,k},
\]

where \(S_{T,P,k}\) is a set of states, and for any \(s \in S_{T,P,k}\), the output \(C(s)\) is a pair that consists of a PRF value and a (new) state. We denote \(C(s) = (C_L(s), C_R(s))\) and define recursively the set of reachable states from a subset \(S\) of \(S_{T,P,k}\) as

\[
\mathcal{R}(S) = S \cup \mathcal{R}(C_R(S)).
\]

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

1. \(\{ \tau \mid \tau \leftarrow T(P, k) \} \cup \{ \perp \} \subseteq \text{St}_{P,k}, \text{ for any } k \in \mathcal{K}\) (initialization).
2. \(C_{R}(\perp) = \perp\) (termination condition).
3. There is a polynomial \(q\) such that for every \(k \in \mathcal{K}\) and \(\tau \leftarrow T(P, k)\):
   
   (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 \(C\). Moreover, starting from an arbitrary trapdoor for \(P\) as initial input, the proxy can execute a number of steps of algorithm \(C\), during which it successively computes the DPRF image of every argument \(x\) that satisfies \(P\), and eventually 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, the completeness condition (item 3.(iii) above) would be infeasible to meet, 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. We consider the case where the proxy is malicious, and accordingly model DPRF security as an oracle indistinguishability game \(G_{\text{sec}}^A\). We now construct \(A'\) as follows out of \(A\): \(A'\) plays the DPRF game and queries the DPRF function for the first \(j\) queries of \(A\). Then it submits the \((j+1)\)-th query of \(A\) as the challenge. Finally, it completes the simulation of \(A\) by answering any remaining queries of \(A\) with random values drawn from \(B\). We show that a good DPRF should allow \(\mathcal{A}'\) to distinguish between two oracles with arbitrarily small probability. Hence, the security of \(G_{\text{sec}}^A\) is more elaborate than the security game of a plain PRF: Indeed, in addition to issuing standard PRF queries, the pseudorandomness attacker may now adaptively query also a trapdoor oracle for delegation on predicates of its choice.

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

\[
\Pr[\mathcal{G}_{\text{sec}}^A(1^\lambda) = 1] \leq \frac{1}{2} + \text{neg}(\lambda).
\]
The existence of a trivial DPRF construction with respect to arbitrary policies from any given PRF motivates our primitive: Interesting DPRF constructions will be those that are \textit{bandwidth efficient}, i.e., they provide trapdoors with size that is \textit{sublinear} in the number of elements that satisfy the corresponding policy predicate.

\textbf{Policy privacy.} We next introduce an additional privacy property that a DPRF scheme may optionally satisfy. This property goes beyond the standard (D)PRF security and is relevant in the context of PRF delegation for protecting the associated policy predicates over which the delegation is defined. We accordingly model this new privacy condition as a \textit{predicate indistinguishability game} \( \mathcal{G}_{\text{pp}} \) carried out between an attacker \( A \) and a challenger \( C \) indexed by a security parameter \( \lambda \). The game proceeds as shown in Figure 3.

\begin{center}
\textbf{DPRF Policy Privacy Game} \( \mathcal{G}_{\text{pp}}(\lambda) \)
\begin{enumerate}
\item The challenger \( C \) randomly selects \( k \) from \( K \).
\item The adversary \( A \) is allowed to interact with \( C \) and ask delegation queries on policy predicates \( P \in \mathcal{P} \); to those queries \( C \) responds with \( \tau \leftarrow T(\mathcal{P}, k) \) and adds \( P \) to a set \( L_{\text{pol}} \).
\item The adversary \( A \) submits two policy predicates \( P_0, P_1 \) to \( C \). The challenger flips a coin \( b \) and computes \( \tau^* \leftarrow T(P_b, k) \). It returns \( \tau^* \) to the adversary.
\item The adversary \( A \) continues as in step 2.
\item The adversary \( A \) terminates by returning a single bit \( b \).
\end{enumerate}
\end{center}

Subsequently the game returns a bit that is 1 if and only if the following holds true:

\[
(b = \hat{b}) \land (|A_{P_0}| = |A_{P_1}|) \land (A_{P_0} \neq A_{P_1}) \land \\
\forall S \subseteq L_{\text{pol}} : \big| \big( \bigwedge_{P \in S} A_{P} \big) \cap A_{P_0} \big| = \big| \big( \bigwedge_{P \in S} A_{P} \big) \cap A_{P_1} \big|. 
\]

Figure 3: The DPRF policy privacy game.

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

\[
\Pr[\mathcal{G}_{\text{pp}}(1^\lambda) = 1] \leq \frac{1}{2} + \negl(\lambda).
\]

The above definition suggests that the trapdoor that corresponds to a certain policy predicate hides the predicate itself (i) at least among all (distinct) policy predicates that enable evaluations of the PRF on the same number of elements in its domain, 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 in the definition are necessary: If \(|A_{P_0}| \neq |A_{P_1}|\), then the adversary can distinguish \( P_0 \) from \( P_1 \) by counting the number \(|\{|(C_{L}(s) \mid s \in \mathcal{R}(\tau^*))\} \mid \forall P \subseteq \mathcal{P} \}| \). In addition, if the number of elements that satisfy simultaneously any set \( S \) of 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 \( \{|(C_{L}(s) \mid s \in \mathcal{R}(\tau^*)) \cap \bigwedge_{P \subseteq S} B_{P}\} \).

We note that while the above formulation can capture a wide set of attacks against privacy, it can be strengthened further by allowing \textit{multiple} challenge pairs of policy predicates. In this case, the policy-privacy 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 \textit{multiple-challenge policy privacy} formulation can be shown to be strictly stronger than the one corresponding to Definition 4.

\textbf{On efficiently achieving policy privacy.} 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 \([0, 2^n - 1]\), where \( n \) is a positive integer. Then, no efficient and policy-private DPRF scheme exists, if the trapdoor-generation algorithm \( T \) is \textit{deterministic}, and the delegated-computation algorithm \( C \) is \textit{deterministic and tree-wise}. This means that for each range, the trapdoor consists of a data structure of initial keys and meta-data that enable in a deterministic way the calculation of the final set of PRF values through a tree-like derivation process, where the final underlying tree structure of this process depends only on the meta-data (and not on any of the initial keys). Indeed, if policy privacy is satisfied, then for every \( 0 < j \leq b - a \), the delegated computation of the PRF values in the intersection \([a + j, b]\) must be indistinguishable from \([a, b]\) 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 \( d_1, \ldots, d_j \) 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 that 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 PRF values computed are at the leaves of a full binary tree, and the policy predicates are ranges of the form \([a, b]\), covered by one or more proper subtrees. In this case, even allowing a proba-
bilistric 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 \(\tau\) be a trapdoor for \([a, b]\)
of size \(g(r) = O(\log^2 r)\), i.e., \(g(r)\) is polylogarithmic in \(r\), hence
also in \(\lambda\). By the Pigeonhole principle, there exists a delega-
tion key \(y\) in \(\tau\) 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 \(d \geq \log(r) - \log(g(r)) = \omega(1)\). 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 \(y\). (Here, a
tree node contributes a 0 or 1 to root-to-leaves path labelings
if it is a left or right child respectively). Since there are no
more than \(g(r)\) such possible values \(x \in [a, b]\), an adversary
has significant probability greater than \(1/g(r)\) of guessing \(x\).
Then it can make a query \(x\), receive \(f_\lambda(x)\) 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 \(\tau^*\), 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 \(f_\lambda(x)\). 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 ex-
tended 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 next.

\textbf{Union policy privacy.} We finally introduce the notion
of \textit{union policy privacy}, where the adversary is restricted
from making queries in the \textit{union} of the challenge policy
predicates (but is allowed to query at arbitrary locations
outside the targeted policy set). This privacy condition is
a strict relaxation of the one corresponding to Definition 4,
and we model it by a game \(G_{\text{UPP}}(1^\lambda)\) that proceeds iden-
tically as \(G_{\text{UPP}}(1^\lambda)\), but terminates with 1 provided that the
following (weaker set of) conditions are all met: (i) \(b \geq \bar{b}\),
(ii) \(|A_{P_1}| = |A_{P_0}|\), (iii) \(A_{P_0} \neq A_{P_1}\) and (iv) \(\forall P \in L_{\text{pol}}:
A_P \cap (A_{P_0} \cup A_{P_1}) = \emptyset\). To see the connection with Defini-
tion 4 observe that the latter condition is equivalent to
\[
\forall S \subseteq L_{\text{pol}}: \left| \left( \bigcap_{P \in S} A_P \right) \cap A_{P_0} \right| = \left| \left( \bigcap_{P \in S} A_P \right) \cap A_{P_1} \right| = 0.
\]
Note that for polices \(P\) consisting of disjoint predicates, the
games \(G_{\text{UPP}}(1^\lambda)\) and \(G_{\text{UPP}}(1^\lambda)\) are equivalent.

\section{Constructions}

In this section we present DPRF schemes for range policy
predicates. In Section 4.1 we describe a first construction,
called \textit{best range cover} (BRC), which satisfies the correct-
ness and security properties of DPRFs, achieving trapdoors
of logarithmic size in the range size. However, BRC lacks the
policy-privacy property. In Section 4.2 we build upon
BRC to obtain a DPRF scheme, called \textit{uniform range cover}
(URC), that is additionally (union) policy private, while
retaining 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.

\subsection{The BRC Construction}

Let \(G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2k}\) be a pseudorandom gen-
erator 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 \(\delta\) is a
random fixed seed (of bit length \(\lambda\)). The GGM pseudorandom
function family \cite{GGM} is defined as \(F = \{f_\lambda : \{0, 1\}^\lambda \rightarrow
\{0, 1\}^\lambda\}_k\), such that
\[
f_\lambda(x_0, \ldots, x_n) = G_{x_0} \cdots (G_{x_n-1}(k)),
\]
where \(n\) is polynomial in \(\lambda\) and \(x_0, \ldots, x_n\) is the input bit-
string of size \(n\).

The GGM construction defines a binary tree over the PRF
domain. We illustrate this using Figure 4, which depicts a
binary tree with 5 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 \textit{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\}\. Then, the PRF value of 0010 is
\(f_\lambda(0010) = G_0(G_1(G_0(k)))\). Observe that the com-
position of \(G\) is performed according to the edge labels in the
path from the root to leaf 2 = (0010)\textsubscript{2}, selecting the first
(second) half of the output of \(G\) when the label of the vis-
ited edge is 0 (resp. 1) and using this half as the seed for
the next application of \(G\). 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—these values constitute the PRF range.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{ggm_tree}
\caption{A GGM tree example.}
\end{figure}

\begin{itemize}
\item Note that we can also associate every \textit{internal} node of the GGM tree with a \textit{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 4 is
associated with partial PRF \(G_0(k)\). Henceforth, for
simplicity, we denote by \(f_\lambda(x_{n-1}, \ldots, x_j)\) the partial PRF
\(G_{x_j, \ldots, x_{n-1}}(k)\). Observe that if a party has the partial
PRF \(f_\lambda(x_{n-1}, \ldots, x_j)\), then it can compute the PRF values of
all \(2^j\) inputs that have prefix \(x_{n-1}, \ldots, x_j\), simply by following
a DFS traversal in the subtree rooted at (the node labelled
by) \(x_{n-1}, \ldots, x_j\) and composing with seed \(f_\lambda(x_{n-1}, \ldots, x_j)\). In
our running example, using the partial PRF value at node
00, we can derive the PRF values of the inputs in the \(\text{dec-
imal}\) range \(0,3\) as \(f_\lambda(0000) = G_0(G_1(f_\lambda(00))))\),
\(f_\lambda(0001) = G_1(G_0(f_\lambda(00))))\), \(f_\lambda(0010) = G_0(G_1(f_\lambda(00))))\),
\(f_\lambda(0011) = G_1(G_0(f_\lambda(00))))\).
\item For any range \([a, b]\) of leaf labels, there is a (non-unique)
set of subtrees in the GGM tree that \textit{cover} exactly the corre-
spending leaves. For instance, \([2, 7]\) is covered by the sub-
trees rooted at nodes 001 and 01 (colored in grey). Ac-
\end{itemize}
The Trapdoor Generation Algorithm \(T_{\text{BRC}}\)

Input: \(a, b: 0 \leq a < b \leq a + \lambda \leq 2^n - 1\) and \(k \in \{0, 1\}^3\)

Output: Trapdoor \(\tau\) for computing \(f_k(x)|x \in [a, b]\)

1. \(\tau \leftarrow \{\}\)
2. \(t \leftarrow \max\{i | a_i \neq b_i\}\)
3. if \(\forall i: t < a_i\) then
4. \(i \leftarrow \min\{i | i < t \land a_i = 1\}\)
5. for \(i = t - 1\) to \(k + 1\)
6. if \(a_i = 0\) then
7. \(\tau \leftarrow \langle f_k(a_{n-1} \ldots a_{t+1}), t \rangle \) to \(\tau\)
8. \(\mu \leftarrow \min\{i | i < t \land a_i = 1\}\)
9. for \(i = t - 1\) to \(\mu + 1\)
10. if \(a_i = 0\) then
11. \(\tau \leftarrow \langle f_k(a_{n-1} \ldots a_{i+1}), t \rangle \) to \(\tau\)
12. \(\tau \leftarrow \langle f_k(a_{n-1} \ldots a_{i}), t \rangle \) to \(\tau\)
13. if \(\forall i: t < b_i\) then
14. \(i \leftarrow \min\{i | i < t \land b_i = 0\}\)
15. for \(i = t - 1\) to \(\nu + 1\)
16. if \(b_i = 0\) then
17. \(\tau \leftarrow \langle f_k(b_{n-1} \ldots b_{i+1}), 0 \rangle \) to \(\tau\)
18. \(\tau \leftarrow \langle f_k(b_{n-1} \ldots b_{i}), 0 \rangle \) to \(\tau\)
19. \(\tau \leftarrow \langle f_k(b_{n-1} \ldots b_{i+1}), 0 \rangle \) to \(\tau\)
20. return \(\tau\)

This algorithm takes as input a secret key \(k\) and a range predicate \([a, b] \in \mathcal{P}\). It outputs a delegation trapdoor \(\tau\) 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 \([a, b]\)). \(T_{\text{BRC}}\) 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 \(T_{\text{BRC}}\) 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, \(T_{\text{BRC}}\) traverses the left and right subtree of \(u\) separately (Lines 7-13 and 14-21, respectively). We will describe only the left traversal (the right one is performed symmetrically). 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 \(r = b - a + 1 = |A_P|\), it finds the minimum number of subtrees that cover \([a, b]\). As such, we call this scheme as best range cover (BRC). A formal description follows.

The BRC DPRF construction is a triplet \((F, T, C)\), where \(F\) is the GGM PRF family described above with tree depth \(n\). The delegation policy is \(\mathcal{P} = \{(a, b) \mid 0 \leq a \leq b \leq a + \lambda \leq 2^n - 1\}\), where \(\gamma\) is a constant integer. Note that when \(a = b\) the trapdoor sent to the proxy is (essentially) 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, henceforth denoted by \(T_{\text{BRC}}\), is given below.

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

**Proof.** We show that all conditions of Definition 2 hold. Condition 1 can be easily seen to hold by the definition of the algorithm, and any instance of algorithm \(T_{\text{BRC}}\) outputs a single pair consisting of the partial PRF value associated with the root of a GGM subtree. Therefore, there is a natural ordering of PRF values for a given \(\tau\), starting from the leftmost leaf of the subtree of \(y_1\) to the rightmost leaf of the subtree of \(y_n\). 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 subtree, the trapdoor \(T\) computes in every next step the PRF value of the next leaf in the ordering discussed above.

In the example of Figure 4, for input \([2, 7]\), \(T_{\text{BRC}}\) outputs trapdoor \(\tau = \langle (f_k(001), 1), (f_k(01), 2)\rangle\). By construction, it is easy to see that the algorithm covers the input range with maximal subtrees, thus using a minimum number of them.
set of states $St_{P,k}$. Moreover, since $R(r) = St_{P,k}$, conditions 2, 3.(i) and 3.(ii) are directly satisfied by the construction 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 $λ^r$ which is a polynomial.

Therefore, it remains to show that condition 3.(iii) holds. Note that, by construction, algorithm $C$ computes exactly the PRF values of the leaves covered by the subtrees corresponding to the partial PRF values included in $r$ by algorithm $T_{BRC}$. Hence, it suffices to show that the labels of these leaves comprise exactly the set of values in $[a, b]$. Let $t = \max\{i \mid a_i ≠ b_i\}$. 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 $T_{BRC}$, 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

$$V_1 = \{x \in [0, 2^t - 1] \mid x_{a_1} \cdots x_{a_t} = a_{a_1} \cdots a_t \wedge (\exists i < t : x_i = 1 \wedge a_i = 0 \vee x = a)\}$$

$$= \{x \in [0, 2^t - 1] \mid a ≤ x ≤ a_{a_1} \cdots a_{a_t} \cdot 1 \cdots 1\}$$

Similarly, we get that $V_2 = [b_{a_1} \cdots b_t 0 \cdots 0 b]$. Observe that, by the definition of $t$, $a_1 = 0$ and $b_1 = 1$. Thus, it holds that $a_{a_1} \cdots a_{a_t} 1 \cdots 1 = b_{a_1} \cdots b_t 0 \cdots 0 b$. This means that $[a, b] = V_1 \cup V_2$, which concludes our proof. $\square$

We next discuss the trapdoor size complexity in BRC. Let $V_1, V_2$ be defined as in the proof of Theorem 1. Then, $|V_1| + |V_2| = r$. We will analyze $|V_1|$ (the analysis for $V_2$ is similar). Observe that, in every step, Lines 3-13, the algorithm covers more than $|V_1|/2$ 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 $r$ is $O(\log(r))$.

We have explained the correctness of BRC and its efficient trapdoor size. We will also prove its security in Section 4.3. Nevertheless, BRC does not satisfy policy privacy, even for non-intersecting policy predicates. We illustrate this with an example using the tree of Figure 4. Consider the ranges $[2, 7]$ and $[9, 14]$, both with size 6, which generate trapdoors $\langle((f_k(001), 1), (f_k(01), 2))\rangle$ and $\langle((f_k(101), 1), (f_k(001), 0), (f_k(110), 1), (f_k(111), 0))\rangle$, respectively; these are trivially distinguishable due to their different sizes. This motivates our second DPRF construction presented next.

### 4.2 The URC Construction

Consider again the ranges $[2, 7]$ and $[9, 14]$ in the tree of Figure 4, for which BRC produces two distinguishable trapdoors, namely trapdoor $\langle((f_k(001), 1), (f_k(01), 2))\rangle$ and trapdoor $\langle((f_k(101), 1), (f_k(001), 0), (f_k(110), 1), (f_k(111), 0))\rangle$. Now, instead of computing the trapdoor of $[2, 7]$ as above, assume that we generate an alternative trapdoor equal to $\langle((f_k(010), 1), (f_k(0001), 0), (f_k(011), 1), (f_k(011), 0))\rangle$. 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** For an integer $r > 1$, a pair of non-negative integral sequences $D = \langle(k_1, \ldots, k_r), (l_1, \ldots, l_r)\rangle$ is called a decomposition of $r$, if the following hold:

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

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

A decomposition of $r$ is $D = \langle(k_1, \ldots, k_r), (l_1, \ldots, l_r)\rangle$ is a worst-case decomposition (w.c.d.) of $r$ if it is of maximum size, i.e., for every decomposition of $r$ $D' = \langle(k_1', \ldots, k_r'), (l_1', \ldots, l_r')\rangle$, we have that $c + d' ≤ c + d$. We define $\Delta_D ≡ \max\{k_1, l_1\}$ as the maximum integer that appears in $D$.

By the description of algorithm $T_{BRC}$ 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 next lemma shows that the maximum integer that appears in any decomposition of $r$ (hence, the maximum depth of a subtree in a cover of a range of size $r$), can only take on two of two consecutive values that depend only on $r$.

**Lemma 1** Let $D = \langle(k_1, \ldots, k_λ), (l_1, \ldots, l_λ)\rangle$ be a decomposition of $r$. Define $B(r) ≡ |\log(r + 2)| - 2$. 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

$$r = \sum_{i=1}^c 2^{k_i} + \sum_{j=1}^d 2^j ≤ 2^{k_{i+1}} + 2^{l_{j+1}} - 2 ≤ \sum_{i=1}^{2M+2} 2^{i+2} - 2 \Rightarrow 2^{M+2} ≥ r + 2 \Rightarrow M_D ≥ B(r).$$

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

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 $\lceil|0, \ldots, B(r)| + |(0, \ldots, B(r) + 1)|\rceil = 2B(r) + 3$ pairs. Hence, the trapdoor size is $O(\log(r))$, which complies with the bound we described in Section 4.1. Moreover, since $|A_P| ≤ λ^r$, every trapdoor has no more than $2[\log(λ^r + 2)] - 1$ 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, hereafter referred to as \( T_{\text{URC}} \), is to construct a converter that takes as input a BRC trapdoor, and produces an alternative trapdoor that complies with a fixed worst-case decomposition. Before proceeding to the details of \( T_{\text{URC}} \), we must prove the following vital theorem.

**Theorem 2** Let \( D = \langle (k_1, \ldots, k_r), (l_1, \ldots, l_d) \rangle \) 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 \( x > 0 \) be the maximum integer in \( \{0, \ldots, M_D\} \) that does not appear in \( D \). Since \( x < M_D \), \( x+1 \) must appear in \( D \). Assume w.l.o.g. that \( l_i = x+1 \). If \( x > k_i \), then the decomposition of \( r \), \( \langle (x, k_1, \ldots, k_d), (l_1, \ldots, l_{i-1}, x, l_{i+1}, \ldots, l_d) \rangle \), is of greater size than that of \( D \). Similarly, if \( i = \min \{i \mid x < k_i\} \), then the decomposition of \( r \), \( \langle (k_1, \ldots, k_i, x, k_{i+1}, \ldots, k_d), (l_1, \ldots, l_{i-1}, x, l_{i+1}, \ldots, l_d) \rangle \), is of greater size than that of \( D \). Both cases contradict the hypothesis, hence, \( x \) must appear in \( D \).

For the converse, let \( D' = \langle ((k_1', \ldots, k_r'), (l_1', \ldots, l_d') \rangle \) 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 \( y_1 \geq \ldots \geq y_{\rho} \) and \( z_1 \geq \ldots \geq z_{\rho} \), respectively. Since an integer cannot appear more than once in a decomposition of \( r \) and, by Lemma 1, the maximum possible value \( B(r) + 1 \) cannot appear more than once, we have that \( y_1, \ldots, y_{\rho} \) and \( z_1, \ldots, z_{\rho} \) are sequences of distinct integers such that \( \sum_{i=1}^{\rho} y_i = \sum_{i=1}^{\rho} z_i \). Assume that there exists a minimum index \( \rho \leq s \) such that \( y_s \neq z_s \) and that w.l.o.g. \( y_\rho > z_\rho \). Then we have the contradiction:

\[
\sum_{i=1}^{s} y_i \geq \sum_{i<s} y_i + y_\rho > \sum_{i<s} y_i + 2^\rho - 1 \geq \\
> \sum_{i<s} 2^i + \sum_{i\geq s} 2^i = \sum_{i=1}^{\rho} y_i.
\]

Thus, \( \{y_1, \ldots, y_{\rho}\} \) and \( \{z_1, \ldots, z_{\rho}\} \) are equal, and therefore \( D \) is a w.c.d. of \( r \). \( \square \)

A consequence of Theorem 2 and Lemma 1 is that, for every integer \( r > 1 \), a w.c.d. of \( r \) is a proper re-arrangement 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 \( T_{\text{URC}} \) in URC, whose pseudocode is provided below. The process starts with invoking the \( T_{\text{BRC}} \) algorithm of BRC to get an initial trapdoor \( \tau \) (Line 1). If the input range is covered by a single subtree, then \( \tau \) is reformed into two pairs that correspond to the child subtrees, so that the depths in \( \tau \) always form a decomposition (Lines 2-3). Let \( D \) be the decomposition implied by \( \tau \) (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 \( \tau \) 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 6). 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 \( \tau \), then the trapdoor can still produce the same leaf PRF values as before. However, at all times we wish the sequence of depths in \( \tau \) 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 \( \tau \) (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 terminates, after properly re-arranging the elements of \( \tau \), 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.

**The Trapdoor Generation Algorithm \( T_{\text{URC}} \)**

**Input:** \( a, b : 0 \leq a < b \leq a + \lambda + 2^\rho - 1 \) and \( k \in \{0, b\}^* \)

**Output:** Trapdoor \( \tau \) for computing \( \langle f(x) | x \in [a, b] \rangle \)

1. Invoke \( T_{\text{BRC}}(a, b, k) \) and receive the output \( \tau \)
2. if \( \tau \) consists only of pair \((y, d)\) then
3. \( \tau \leftarrow \langle (G_0(y), d-1), (G_1(y), d-1) \rangle \)
4. Let \( \tau = \langle (y_1, d_1), \ldots, (y_\rho, d_\rho) \rangle \) and \( D = \langle (d_1, \ldots, d_\rho), (d_{\rho+1}, \ldots, d_m) \rangle \)
be the corresponding 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. 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. Remove \((y_i, x+1)\) from \( \tau \), insert the pairs
\((y_i^0, x)\) and \((y_i^1, x)\) in \( \tau \) respecting the strictly decreasing order, and update \( D \) accordingly
8. if the leftmost sequence of \( D \) is not \((B(r), \ldots, 0)\) then
9. Fill leftmost sequence with values from rightmost sequence until it complies with the uniform decomposition of \( r \)
10. return \( \tau \)

In our running example, for the range \([2, 7]\), the \( T_{\text{URC}} \) algorithm in URC converts the original token retrieved by the trapdoor algorithm of BRC, \( \tau = \langle (f_0(001), 1), (f_0(01), 2) \rangle \), as follows (we underline a newly inserted element to the left sequence, and depict as bold a newly inserted element to the right sequence):

\[
\langle (f_0(001), 1), (f_0(01), 2) \rangle \\
\langle (f_0(0010), 0), (f_0(01), 2), (f_0(011), 0) \rangle \\
\langle (f_0(010), 1), (f_0(010), 0), (f_0(011), 1), (f_0(0011), 0) \rangle
\]

The \( C \) algorithm of URC is identical to that of BRC, because the trapdoor in URC has exactly the format expected by this algorithm, i.e., pairs of PRF values corresponding to GCM subtrees along with their depths. Moreover, during the evolution of the initial BRC trapdoor into one 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 delegation 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 $r$ 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 range size at most $\lambda^2$ is subtree secure against PPT adversaries that make all their queries, even the challenge query, non-adaptively.

Proof. Let $A$ be a non-adaptive prefix-only PPT adversary against a BRC scheme with depth $n$ and maximum range size $\lambda^2$. Without loss of generality we assume that $A$ always advances to step 3 (submits a challenge to the challenger).

We define recursively two sequences of PPT algorithms $A = A_n, \ldots, A_1$ and $S_n, \ldots, S_1$ as follows.

For $i = n, \ldots, 1$, $A_i$ on input $1^k$, initially invokes $A_{i+1}$ receiving all of its non-adaptive queries, and chooses a random value $k' \in \{0,1\}^k$. If a query $x_1 \ldots x_i$ has the same most significant bit (MSB) as the challenge query, $A_i$ makes the query $x_{i-1} \ldots x_i$, and responds with the received value $y$. Otherwise, it responds with the value $G_{\lambda^2}(\ldots(G_{\lambda^2}(k')))$. It returns $A_{i+1}$’s output. For $i = n, \ldots, 1$, on input $(z_0, z_1) \in \{0,1\}^{2\lambda}$, $S_i$ invokes $A_i$ and receives all of its queries. For every query $x_{i-1} \ldots x_i$, it responds with $G_{\lambda^2}(\ldots(G_{\lambda^2}(z_{i-1})))$ (or $z_0$ for $i = 1$). On the challenge phase, it flips a coin $b$ and acts as the challenger in a DPRF security game with $A_i$. It returns 1 iff $A_i$ returns $b$.

Let $q_i$ and $p_i$ be the probabilities 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 $S$, $p_i = Pr[G_{\lambda^2}(1^\lambda) = 1]$ while $q_i \leq 1/2$, since it corresponds to a totally random game.

We next observe that $A_n, \ldots, A_1$ 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. Finally:

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

hence $Pr[G_{\lambda^2}(1^\lambda) = 1] = p_n \leq q_1 + n \cdot \epsilon(\lambda) \leq 1/2 + n \cdot \epsilon(\lambda)$, where $\epsilon(\lambda) = \max_i \{\epsilon_i(\lambda)\}$. $\square$

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 $\lambda^2$ is subtree secure if $2^{n-1} \leq \lambda^2 < 2^n$.

Proof. Let $A$ be a prefix-only adversary. We construct a non-adaptive prefix-only adversary $B$ for the subtree security game that on input $1^\lambda$ 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, $B$ makes queries $(x_{n-1}^* \oplus 1), \ldots, (x_{i-1}^* \oplus 1), \ldots, (x_{0}^* \oplus 1)$ 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 $A$ and plays the security game with $A$, as a challenger that can respond appropriately for every value that is not $x^*$ or a range that does not contain $x^*$. If $A$ sets $x^*$ as a challenge, then $B$ responds with $y^*$, and returns $A$’s guess. Otherwise, $A$ has either made a query which is a prefix of $x^*$, or it has submitted a challenge different than $x^*$, so $B$ terminates the game it plays with $A$ and returns a random bit, as its guess to its challenge.

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

$$Pr[G_{\lambda^2}(1^\lambda) = 1 \land \neg E] = 1/2 \cdot (1 - 1/2^n) \quad \text{and} \quad Pr[G_{\lambda^2}(1^\lambda) = 1 \land E] = 1/2^n \cdot Pr[G_{\lambda^2}(1^\lambda) = 1].$$

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

$$1/2^n \cdot Pr[G_{\lambda^2}(1^\lambda) = 1] - 1/2 + 1/2 \leq 1/2 + \epsilon(\lambda)\ ,$$

so, $Pr[G_{\lambda^2}(1^\lambda) = 1] = 1/2 + 2^n \cdot \epsilon(\lambda) \leq 1/2 + 2\lambda^2 \cdot \epsilon(\lambda)$. $\square$

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

Lemma 4 The BRC scheme with depth $n$ and maximum range size $\lambda^2$ is subtree secure.

Proof. See Appendix. $\square$

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 $\lambda^2$ are secure.

Proof. Let $A$ be a PPT adversary that wins the DPRF security game of either BRC or URC with non-negligible advantage $\alpha(\cdot)$. As not in the beginning of this section, any query $A$ makes, can be represented as a sequence of $O(\log(\lambda^2))$ subtrees, or equivalently of $O(\log(\lambda^2))$ prefixes. Thus, we can construct a prefix-only adversary $A'$ that invokes $A$ and when it receives a query sequence $(x_{n-1}^* \ldots x_1^*)$,
...., x_{n-1}m, x_m) from A, it makes all prefix queries separately, receives y_1, ..., y_m and answers by (y_1, ..., y_m). A' also transfers A's challenge and outputs its guess. Therefore, it wins the DPRF security game with advantage α(·), 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 ((y_1, 1), (y_2, 0), (y_3, 0)) 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^n is a DPRF with union policy privacy.

**Proof.** See Appendix.

5. Applications

In this section we discuss interesting applications of the general DPRF primitive and our specialized constructions for range-based policies. 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, 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, authentication and access control in RFID-based systems has been studied in the literature. 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 (F, T, C) is directly applicable to the above setting. 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-based 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 bandwidth-efficient tag key delegation from the TC to the reader. Moreover, policy-private DPRFs provide stronger protection for the tag IDs against the readers.

**Batch queries in searchable symmetric encryption.** Searchable symmetric encryption (SSE) (e.g., [11, 24]) enables processing queries directly on ciphertexts generated with symmetric encryption. Although SSE corresponds to 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 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, ..., c_n 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, ..., 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 \([1K, 2K]\), \([2K, 3K]\), \ldots , \([9K, 10K]\), which cover the query range \([1K, 10K]\). Such scenarios are handled with “traditional” SSE as shown in Figure 5(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. These trapdoors are sent to the server, which then searches \(I\) with every \(\tau_{w_i}\) following the SSE protocol. Obviously, for large \(N\), the client’s computational and communication cost is greatly impacted.

A DPRF with multi-instance policy privacy, it is easy to show that the same level of keyword privacy as in [11] can be achieved. 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 delegation queries not intersecting with previously asked queries. Therefore, combining DPRFs 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 levels 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 (e.g., [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 \(\Phi\) 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 \(S_u\).

We can augment the discussed SSE framework with DPRF functionality, in order to support batch queries with sublinear \((N)\) processing and communication cost at the client, while providing provable security along the lines of [11]. Figure 5(b) illustrates our enhanced SSE architecture. Recall that \(\tau_{w_i}\) is practically a PRF value, computed on \(w_i\) with key \(k\). Therefore, instead of computing a PRF value for every \(w_i\) itself, the client delegates the computation of these PRF values to the server by employing a DPRF scheme \((F, T, C)\), where \(F\) is defined over the domain of the keywords. Given predicate \(P\) and \(k\), the client runs \(T\) and generates trapdoor \(\tau\), which is sent to the server. The latter executes \(C\) on \(\tau\) 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
over \( \Phi \). Interestingly, the reverse is also true and a broadcast encryption scheme with efficient key assignment will provide a DPRF with domain \( \Phi \) and the policy set \( \{ S_u | u \in [n] \} \).

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 has been considered in [3, 13, 25] 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’s 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 broadcast encryption—note that none of the tree-like key-delegation methods used in broadcast encryption schemes prior to our work satisfies 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, which prevents leakage of the secret preimages of the computed PRF values. Moreover, we presented two DPRF constructions for PRF delegation controlled by range-based policies, along with a comprehensive analysis in terms of their security and privacy guarantees and some inherent trade-offs with efficiency. Our proposed DPRF schemes 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 key-derivation settings providing interesting new results.

Further exploration of DPRFs holds promise for many directions of interest. 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.

Acknowledgments

We thank all anonymous reviewers for providing detailed comments and suggestions. The first and fourth authors were supported by projects CODAMODA of the European Research Council, Marie Curie RECUP, and respectively FINER of GSRT.

7. REFERENCES


APPENDIX
Proof of Lemma 4

By Lemma 3, it suffices to show for the case that \( \lambda^3 < 2^{n-1} \).

This is definitely the interesting case, since \( 2^n \) is normally superpolynomial in \( \lambda \). Let \( A \) be a prefix-only adversary against BRC that makes at most \( q(\lambda) \) queries (including the challenge) and \( d \) be the minimum integer that \( \lambda^{3d} < 2^d \).

We construct a PPT PRF distinguisher \( B \) that makes oracle queries of fixed size \( n' = n - d \geq 1 \). On input \( 1^\lambda \), \( B \) flips a coin \( b \), invokes \( A \) and initializes a security game, itself being the challenger. By the bound on the size of the ranges, all queries of \( A \) have length greater than \( n' \). Thus, for every query \( x_{n-1} \cdots x_1 \), we have that \( t < d \) and \( B \) responds by making a query \( x_{n-1} \cdots x_d \) which is of length \( n' \), receiving a value \( y \) and answering to \( A \) as \( G_x(\cdot (G_{d-1}(y))) \). When \( A \) submits a challenge, \( B \) 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, \( B \) returns 1 if \( A \) returns \( b \). Clearly, when \( B \)'s oracle is a PRF \( f_k \) of length \( n' \), \( B \) returns 1 iff \( A \)'s wins, i.e \( \Pr_{\beta \in \{0,1\}}[B^{\beta}(1^\lambda)] = \Pr_{B^{\lambda}(1^\lambda)}[B^{k_\lambda}(1^\lambda)] = 1 \). Since \( f_k \) is a PRF, we have that for some negligible function \( \epsilon(\cdot) \),

\[
\Pr_{\beta \in \{0,1\}}[B^{\beta}(1^\lambda)] - \Pr_{A^{\lambda}}[B^{A}(1^\lambda)] = \epsilon(\lambda). \tag{1}
\]

Consequently, we can construct a prefix-only adversary \( \hat{A} \) against a BRC with depth \( d \) and maximum range size \( \lambda^3 \) as follows: \( \hat{A} \) invokes \( A \) and chooses a random index \( j \in [q(\lambda)] \) and \( q(\lambda) \) random values \( k_1, \ldots, k_{j-1}, k_{j+1}, \ldots, k_{q(\lambda)} \in \{0,1\}^{\lambda} \).

Index \( j \) reflects \( \hat{A} \)'s attempt to guess which of all of possible different prefixes of length \( n' \) that will appear in \( A \)'s queries will be the one that a prospective challenge query will have. Then \( \hat{A} \) keeps count of the different prefixes of length \( n' \) that gradually appear and reacts to a query \( x_{n-1} \cdots x_1 \) according to the following checks:

(i). \( x_{n-1} \cdots x_d \) is the \( i \)-th prefix and \( i \neq j \) : In this case, \( \hat{A} \) responds with \( G_x(\cdot (G_{d-1}(k_i))) \).

(ii). \( x_{n-1} \cdots x_d \) is the \( j \)-th prefix: If all of the queries made by \( A \) 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 \( \hat{A} \) terminates the game with \( A \) and chooses a leaf \( z_{d-1} \cdots z_0 \) of \( T_j \) that has not been covered by \( A \)'s previous queries. It submits \( z_{d-1} \cdots z_0 \) as its challenge and returns a random bit. Otherwise, \( \hat{A} \) 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 \( A \)'s challenge: If \( x_{n-1} \cdots x_0 \) is not the \( j \)-th prefix, then \( \hat{A} \) terminates the game with \( A \), chooses a leaf \( z_{d-1} \cdots z_0 \) of \( T_j \) not yet covered by \( A \)'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 \( \hat{A} \) does not terminate the security game with \( A \), then it returns \( A \)'s guess. By the choice of \( d \), \( 2^{d-1} \leq \lambda^3 < 2^d \), hence Lemma 3 implies that \( \hat{A} \) has negligible distinguishing advantage. Thus for some negligible function \( \delta(\cdot) \),

\[
\Pr_{A^{\lambda}}[B^{A}(1^\lambda)] \leq 1/2 + 2\delta(\lambda). \tag{2}
\]
By the description of $\tilde{A}$, the interaction between $A$ and $B$ is in the case that $B$’s oracle is random is fully simulated by $\tilde{A}$ when the latter’s guess for the prefix of $\tilde{A}$’s challenge is correct. Formally, let $E$ be the event that $\tilde{A}$ guesses $A$’s challenge. Then it holds that

$$\Pr[B^{R_C}(1^\lambda) = 1] = \Pr[G^A_{\text{SEC}}(1^\lambda) = 1/|E|].$$

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

$$\Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land \neg E] = 1/2 \cdot (1 - 1/q(\lambda)) \text{ and } \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land E] = 1/q(\lambda) \cdot \Pr[B^{R_C}(1^\lambda) = 1],$$

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

$$\Pr[G^A_{\text{SEC}}(1^\lambda) = 1] = 1/2 + 1/q(\lambda) \cdot \Pr[B^{R_C}(1^\lambda) = 1] - 1/2,$$

so by applying (2) we get

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

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

$$\Pr[G^A_{\text{SEC}}(1^\lambda) = 1] \leq 1/2 + q(\lambda) \cdot \delta(\lambda) + \epsilon(\lambda) = 1/2 + \text{neg}(\lambda).$$

**Proof of Theorem 4**

Let $A$ be a PPT adversary that wins the union policy privacy game with non-negligible distinguishing advantage $\alpha(\cdot)$. Let $P_b$, $P_i$ be the two challenge ranges and $b$ the chosen random bit, hence $A$ receives the challenge trapdoor $\tau_i$ for $P_i$. 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 $\prec$ of the elements of $D$ as follows: $(x, L) \prec (y, R)$ if $(x, R) \prec (y, L)$, if $x < y$ and $(x, L) \prec (x, R)$ if $y < x$. We define a sequence of hybrid games $G^A_0(1^\lambda), \ldots, G^A_N(1^\lambda)$, where $N$ is the maximum size of a trapdoor output by $T_{\text{URC}}$. As shown in section 4.2, $N = 2\lceil\log(\lambda^2 + 2\lambda)\rceil - 1$. The game $G_i$ is executed as the original game $G^A_{\text{GRP}}(1^\lambda)$ during the pre-challenge and the post-challenge phase. Assume the pairs of the challenge trapdoor $\tau_i$ are arranged according to the ordering determined by the decomposition formed by the depths. The only modification in $G^A_0(1^\lambda)$ is that in the first $i$ pairs of $\tau_i$, the partial PRF values are the same as $\tau_i$’s, while all the other elements are replaced by random values in $(0, 1)^\lambda$. In $G^A_i(1^\lambda)$, the challenge trapdoor consists of a number of pairs of random values attached to certain integers, independently of the choice of $b$. Therefore, $\Pr[G^A_0(1^\lambda) = 1] = 1/2$. Since $G^A_0(1^\lambda)$ is the UPP game, it holds that

$$\Pr[G^A_0(1^\lambda) = 1] - \Pr[G^A_0(1^\lambda) = 1] \geq \alpha(\lambda).$$

Let $E_i$ be the event that the size of the challenge ranges $|A_{P_i}|, |A_{P_i}|$ that $A$ submits is $r$. Then for some challenge bit $b \in \{0, 1\}$ and $r \in [\lambda]$: $G^A_i(1^\lambda) = 1 \land b \land E_i \land A_{P_i} = 1 \land b \land E_i \geq \alpha(\lambda)/2\lambda$, which implies that there exists an $i \in [\lambda]$ such that

$$\Pr[G^A_i(1^\lambda) = 1 \land b \land E_i] - \Pr[G^A_{i-1}(1^\lambda) = 1 \land b \land E_i] \geq \alpha(\lambda)/2N\lambda^\gamma.$$

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

On input $1^\lambda$, $A_i$ computes the uniform decomposition of $r, u_i$, and arranges its elements according to $<_{U_i}$. Let $u_i$ be the integer that appears in the $i$-th element of $U_i$. The BRC that $A_i$ attacks has depth $n - u_i$. Specifically, $A_i$ invokes $A$ and answers all of its pre-challenge queries as follows: for each query $x_{n-1} \cdots x_1$, 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 $G_{x_1}(\cdots(G_{x_{u_i-1}}(y)))$. In the challenge phase, if $|A_{P_b}| \neq r$, then $A_i$ terminates the game with $A$, chooses a valid random challenge, and returns a random bit. Otherwise, it makes $i - 1$ queries and computes the $<_{U_i}$-first $i - 1$ partial delegation keys $y_1, \ldots, y_{i-1}$ of $\tau_i$ as in the pre-challenge phase, and sets the string $x_{n-1} \cdots x_{u_i}$ that corresponds to the $i$-th key as its challenge, receiving $y^*$. Note that since $A$ is restricted from making queries within $A_{P_i} \cup A_{P_j}$, it makes no queries with prefix $x_{n-1} \cdots x_{u_i}$, thus $A_i$’s challenge is valid. It arranges the values $y_1, \ldots, y_{i-1}, y^*$ according to the order that $\tau_i$ imposes and “fills” the “[U_i]” remaining positions of an array like trapdoor $\tau_i$ with $|U_i| - i$ pairs consisting of random values from $(0, 1)^\lambda$ along with the corresponding depths. It returns $\tau_i$ to $A$ and answers to $A$’s post-challenge queries as in the pre-challenge phase. If $A$ returns $b$, then $A_i$ returns 1, or else it returns a random bit.

The probability that $A_i$ wins the security game is

$$\Pr[G^A_{\text{SEC}}(1^\lambda) = 1] = \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land \neg E_i] + \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land E_i \land b_i = 1] + \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land E_i \land b_i = 0].$$

It holds that $\Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land \neg E_i] = 1/2 \cdot (1 - \Pr[E_i])$.

For the other two terms in the right part of (6), we observe that when $b_i = 1$, $A_i$ simulates $G_i$ when $b$ and $E_i$ occur, whereas when $b_i = 0$, $A_i$ simulates $G_{i-1}$ when $b$ and $E_i$ occur. Therefore, by the description of $A_i$ we have that

$$\Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land E_i \land b_i = 1] =$$

$$= 1 \cdot \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land b \land E_i] +$$

$$+ 1/2 \cdot \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land b \land E_i] =$$

$$= 1/2 \cdot \Pr[b \land E_i] + 1/2 \cdot \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land b \land E_i].$$

and

$$\Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land E_i \land b_i = 0] =$$

$$= 0 \cdot \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land b \land E_i] +$$

$$+ 1/2 \cdot \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land b \land E_i] =$$

$$= 1/2 \cdot \Pr[b \land E_i] - 1/2 \cdot \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land b \land E_i].$$

By the independency of $b$ and $E_i$, we have that $\Pr[b \land E_i] = 1/2 \cdot \Pr[E_i]$. Thus, we evaluate $\Pr[G^A_{\text{SEC}}(1^\lambda) = 1]$ according to (6) as

$$\Pr[G^A_{\text{SEC}}(1^\lambda) = 1] = 1/2 + 1/2 \cdot (\Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land b \land E_i] -$$

$$- \Pr[G^A_{\text{SEC}}(1^\lambda) = 1 \land b \land E_i]).$$

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