Constrained PRFs for Unbounded Inputs with Short Keys

Hamza Abusalah[1(✉)] and Georg Fuchsbauer[2]

[1] Institute of Science and Technology Austria, Klosterneuburg, Austria
habusalah@ist.ac.at
[2] Inria, ENS, CNRS and PSL Research University, Paris, France
georg.fuchsbauer@ens.fr

Abstract. A constrained pseudorandom function (CPRF) $F\colon \mathcal{K}\times\mathcal{X}\to\mathcal{Y}$ for a family $\mathcal{T}$ of subsets of $\mathcal{X}$ is a function where for any key $k\in\mathcal{K}$ and set $S\in\mathcal{T}$ one can efficiently compute a short constrained key k_S, which allows to evaluate $F(k,\cdot)$ on all inputs $x\in S$, while the outputs on all inputs $x\notin S$ look random even given k_S.

Abusalah et al. recently constructed the first constrained PRF for inputs of arbitrary length whose sets S are decided by Turing machines. They use their CPRF to build broadcast encryption and the first ID-based non-interactive key exchange for an unbounded number of users. Their constrained keys are obfuscated circuits and are therefore large.

In this work we drastically reduce the key size and define a constrained key for a Turing machine M as a short signature on M. For this, we introduce a new signature primitive with constrained signing keys that let one only sign certain messages, while forging a signature on others is hard even when knowing the coins for key generation.

Keywords: Constrained PRFs · Unbounded inputs

1 Introduction

Constrained PRFs. A pseudorandom function (PRF) [15] is a keyed function $\mathsf{F}\colon \mathcal{K}\times\mathcal{X}\to\mathcal{Y}$ for which no efficient adversary, given access to an oracle $\mathcal{O}(\cdot)$, can decide whether $\mathcal{O}(\cdot)$ is $\mathsf{F}(k,\cdot)$ with a random key $k\in\mathcal{K}$, or whether $\mathcal{O}(\cdot)$ is a uniformly random function $\mathcal{X}\to\mathcal{Y}$. A PRF F is called *constrained* [7,10,17] for a predicate family $\mathcal{P}$ if additionally there exists a PPT constraining algorithm $k_p \leftarrow \mathsf{F.Constr}(k,p)$ that, on input a key k and a predicate $p\colon \mathcal{X}\to\{0,1\}$ specifying a subset $S_p=\{x\in\mathcal{X}\mid p(x)=1\}$ of potentially exponential size, derives a constrained key k_p. The latter allows computing $\mathsf{F}(k,x)$ on all $x\in S_p$, while even given keys for $p_1,\ldots,p_\ell$, values $\mathsf{F}(k,x)$ for $x\notin\bigcup_i S_{p_i}$ still look random. Note that if all sets S_p are polynomial-size, a simple solution would be

H. Abusalah—Research supported by the European Research Council, ERC starting grant (259668-PSPC) and ERC consolidator grant (682815 - TOCNeT).

M. Manulis et al. (Eds.): ACNS 2016, LNCS 9696, pp. 445–463, 2016.
DOI: 10.1007/978-3-319-39555-5_24

to set $k_p := \{\mathsf{F}(k,x) \,|\, x \in S_p\}$, which would achieve the desired security. The challenge is to have short keys for potentially big sets.

The simplest type of constrained PRFs (CPRF) are puncturable PRFs [18], where for any input $x \in \{0,1\}^*$ one can derive a key k_{x^*} that allows evaluation everywhere except on x^*, whose image is pseudorandom even given k_{x^*}. The most general CPRF is one that is constrained w.r.t. a Turing-machine (TM) predicate family $\mathcal{M}$, where $M \in \mathcal{M}$ defines a subset of inputs of arbitrary length $S_M = \{x \in \{0,1\}^* \,|\, M(x) = 1\}$. In a TM-constrained PRF a constrained key k_M can be derived for any set S_M defined by a TM M.

Abusalah et al. (AFP) [2] construct a (selectively secure) TM-constrained PRF and show how to use it to construct broadcast encryption (BE) [8,11] where there is no a priori bound on the number of possible recipients and the first identity-based non-interactive key-exchange scheme [7,12,19] with no a priori bound on the number of parties that agree on a key.

The main shortcoming of their construction is that a constrained key k_M for a TM M is an obfuscated circuit and is therefore not short but typically huge. This translates to large user keys in the BE and ID-NIKE schemes built from their CPRF. In this paper we overcome this and reduce the key size drastically by defining a constrained key k_M for M as simply a signature on M.

TM-Constrained PRFs with Short Keys. The AFP TM-constrained PRF in [2] is built from puncturable PRFs, succinct non-interactive arguments of knowledge (SNARKs), which let one prove knowledge of an NP witness via a short proof, collision-resistant hashing and public-coin differing-input obfuscation ($di\mathcal{O}$) [16]. The latter lets one *obfuscate* programs, so that one can only distinguish obfuscations of two equal-size programs if one knows an input on which those programs' outputs are different. Moreover, if for two programs it is hard to find such a differing input, *even when knowing the coins used to construct the programs*, then their obfuscations are indistinguishable.

Relying on essentially the same assumptions, we enhance the AFP construction and obtain short constrained keys. Let us look at their CPRF F first, which is defined as $\mathsf{F}(k,x) := \mathsf{PF}(k, H(x))$, where PF is a puncturable PRF, and H is a hash function (this way they map unbounded inputs to constant-size inputs for a puncturable PRF). A constrained key for a TM M is a $di\mathcal{O}$ obfuscation of the circuit P_M that on input (h,π) outputs $\mathsf{PF}(k,h)$ iff π is a valid SNARK proving the following statement: $(*)$ $\exists x : h = H(x) \wedge M(x) = 1$. So P_M only outputs the PRF value if the evaluator knows such an input x.

We also define our TM-CPRF as $\mathsf{F}(k,x) := \mathsf{PF}(k, H(x))$. However at setup, we publish as a public parameter once and for all a $di\mathcal{O}$-obfuscated circuit P that on input (h,π,M,σ) outputs $\mathsf{PF}(k,h)$ iff π is a valid SNARK for $(*)$ and additionally σ is a valid signature on M. A constrained key k_M for a TM M is a signature on M and a party holding $k_M := \sigma$ can generate a SNARK π, as in the AFP construction, and additionally use M, σ to run P to evaluate F.

The intuition behind the construction is simple: in order to evaluate F on x, one needs a signature on a machine M with $M(x) = 1$. Unforgeability of such signatures should guarantee that without a key for such an M the PRF value of x should be pseudorandom. However, actually proving this turns out quite tricky.

In the selective security game for CPRFs, the adversary first announces an input x^* and can then query keys for sets that do not contain x^*, that is, sets decided by TMs M with $M(x^*) = 0$. The adversary then needs to distinguish the PRF image of x^* from random. To argue that $\mathsf{F}(k, x^*)$ is pseudorandom, we replace the circuit P by P^* for which F looks random on x^*, because it uses a key that is punctured at $H(x^*)$. Intuitively, since P is obfuscated, an adversary should not notice the difference. However, to formally prove this we need to construct a sampler that constructs P and P^* and argue that it is hard to find a differing input (h, π, M, σ) even when given the coins to construct the circuits.

One such differing input would be one containing a signature $\hat{\sigma}$ on a machine $\hat{M}$ with $\hat{M}(x^*) = 1$. Since $\hat{\sigma}$ is a key for a set containing x^*, P outputs the PRF value, while P^* does not, as its key is punctured. As the adversary only obtains signatures for M's with $M(x^*) = 0$, $\hat{\sigma}$ intuitively is a forgery. But the sampler that computes P and P^* also computed the signature verification key. So how can it be hard to construct a differing input containing $\hat{\sigma}$ for someone knowing the coins that also define the secret key?

We overcome this seeming contradiction by introducing a new primitive called *functional signatures with obliviously samplable keys* (FSwOSK). To produce the circuits P, P^*, the sampler needs to answer the adversary's key queries, that is, compute signatures on M's with $M(x^*) = 0$. FSwOSK lets the sampler create a pair of verification and signing keys, of which the latter only allows to sign such machines M; and security for FSwOSK guarantees that *even when knowing the coins used to set up the keys*, it is hard to create a signature on a message $\hat{M}$ with $\hat{M}(x^*) = 1$.

2 Preliminaries

2.1 Constrained and Puncturable PRFs

Definition 1 (Constrained Functions). *A family of keyed functions* $\mathcal{F}_\lambda = \{F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}\}$ *over a key space* $\mathcal{K}$*, a domain* $\mathcal{X}$ *and a range* $\mathcal{Y}$ *is* efficiently computable *if there exist a probabilistic polynomial-time (PPT) sampler* $\mathsf{F.Smp}$ *and a deterministic PT evaluator* $\mathsf{F.Eval}$ *as follows:*

- $k \leftarrow \mathsf{F.Smp}(1^\lambda)$*: On input a security parameter* λ*,* $\mathsf{F.Smp}$ *outputs a key* $k \in \mathcal{K}$*.*
- $y := \mathsf{F.Eval}(k, x)$*: On input a key* $k \in \mathcal{K}$ *and* $x \in \mathcal{X}$*,* $\mathsf{F.Eval}$ *outputs* $y = F(k, x)$*.*

The family $\mathcal{F}_\lambda$ *is* constrained *w.r.t. a family* $\mathcal{S}_\lambda$ *of subsets of* $\mathcal{X}$*, with constrained key space* $\mathcal{K}_\mathcal{S}$ *such that* $\mathcal{K}_\mathcal{S} \cap \mathcal{K} = \emptyset$*, if* $\mathsf{F.Eval}$ *accepts inputs from* $(\mathcal{K} \cup \mathcal{K}_\mathcal{S}) \times \mathcal{X}$ *and there exists the following PPT algorithm:*

$k_S \leftarrow \mathsf{F.Constr}(k, S)$*: On input a key* $k \in \mathcal{K}$ *and a (short) description of a set* $S \in \mathcal{S}_\lambda$*,* $\mathsf{F.Constr}$ *outputs a constrained key* $k_S \in \mathcal{K}_\mathcal{S}$ *such that*

$$\mathsf{F.Eval}(k_S, x) = \begin{cases} F(k, x) \; if x \in S \\ \perp \qquad otherwise. \end{cases}$$

$\mathbf{Exp}^{\mathcal{O},b}_{\mathcal{F},\mathcal{A}}(\lambda)$:

$k \leftarrow \mathsf{F.Smp}(1^\lambda)$; $C, E := \emptyset$
$(x^*, st) \leftarrow \mathcal{A}_1^{\mathcal{O}_1}(1^\lambda)$
If $x^* \in E$, then abort
If $b = 1$ then $y := \mathsf{F.Eval}(k, x^*)$;
else $y \leftarrow \mathcal{Y}$
$C := C \cup \{x^*\}$
Return $b' \leftarrow \mathcal{A}_2^{\mathcal{O}_2}(st, y)$

Oracle $\textsc{Constr}(S)$:

If $S \notin \mathcal{S}_\lambda \vee S \cap C \neq \emptyset$
  Return $\perp$
$E := E \cup S$
$k_S \leftarrow \mathsf{F.Constr}(k, S)$
Return k_S

Oracle $\textsc{Eval}(x)$:

If $x \notin \mathcal{X} \vee x \in C$
  Return $\perp$
$E := E \cup \{x\}$
$y = \mathsf{F.Eval}(k, x)$
Return y

Fig. 1. The security game for constrained PRFs

Definition 2 (Security of Constrained PRFs). *A family of constrained functions* $\mathcal{F}_\lambda = \{F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}\}$ *is* selectively pseudorandom*, if for every PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *in* $\mathbf{Exp}^{\mathcal{O},b}_{\mathcal{F},\mathcal{A}}$*, defined in Fig. 1, with* $\mathcal{O}_1 = \emptyset$ *and* $\mathcal{O}_2 = \{\textsc{Constr}(\cdot), \textsc{Eval}(\cdot)\}$*, it holds that*

$$\mathsf{Adv}^{\mathcal{O}}_{\mathcal{F},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathbf{Exp}^{\mathcal{O},0}_{\mathcal{F},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathcal{O},1}_{\mathcal{F},\mathcal{A}}(\lambda) = 1\right]\right| = \mathit{negl}(\lambda). \quad (1)$$

Furthermore, $\mathcal{F}_\lambda$ *is* adaptively pseudorandom *if the same holds for* $\mathcal{O}_1 = \mathcal{O}_2 = \{\textsc{Constr}(\cdot), \textsc{Eval}(\cdot)\}$.

Remark 1. We require $\mathbf{Exp}^{\mathcal{O},b}_{\mathcal{F},\mathcal{A}}$ of Fig. 1 to be efficient. Thus when sets are described by Turing machines then every machine M queried to $\textsc{Constr}$ must terminate on x^* within a polynomial number of steps T (as the oracle must check whether $S \cap \{x^*\} \neq \emptyset$, that is, $M(x^*) = 1$).

Puncturable PRFs [18]**.** These are simple constrained PRFs whose domain is $\{0,1\}^n$ for some n and whose constrained keys are for sets $\{\{0,1\}^n \setminus \{x_1, \ldots, x_m\} \mid x_1, \ldots, x_m \in \{0,1\}^n, m = \mathsf{poly}(\lambda)\}$, i.e., a punctured key can evaluate the PRF on all except polynomially many inputs. We only require selective pseudorandomness. A formal definition is given in the full version [1].

Selectively secure puncturable PRFs are easily obtained from selectively secure prefix-constrained PRFs, which were constructed from the GGM PRF [15] in [7,10,17]. In this work we only require selectively secure puncturable PRFs.

2.2 Public-Coin Differing-Input Obfuscation

Public-coin differing-input (di) obfuscation guarantees that if for pairs of *publicly* sampled programs it is hard to find an input on which they differ then their obfuscations are computationally indistinguishable. We follow [16] by first defining public-coin di samplers that output programs whose obfuscations are indistinguishable.

Definition 3 (Public-Coin DI Sampler [16]**).** *A non-uniform PPT sampler* Samp *is a public-coin differing-input sampler for a family of polynomial-size*

circuits $\mathcal{C}_\lambda$ if the output of Samp is in $\mathcal{C}_\lambda \times \mathcal{C}_\lambda$ and for every non-uniform PPT extractor $\mathcal{E}$ it holds that

$$\Pr\left[\begin{array}{l} r \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ (C_0, C_1) := \mathsf{Samp}(1^\lambda, r);\ x \leftarrow \mathcal{E}(1^\lambda, r) \end{array} : \ C_0(x) \neq C_1(x)\right] = \mathsf{negl}(\lambda). \quad (2)$$

Definition 4 (Public-Coin diO [16]). *A uniform PPT algorithm diO is a public-coin differing-input obfuscator for a family of poly-size circuits $\mathcal{C}_\lambda$ if:*

- *For all $\lambda \in \mathbb{N}$, $C \in \mathcal{C}_\lambda$ and x: $\Pr\left[\widetilde{C} \leftarrow \mathsf{diO}(1^\lambda, C) : \ C(x) = \widetilde{C}(x)\right] = 1$.*
- *For every public-coin di sampler Samp for a family of poly-size circuits $\mathcal{C}_\lambda$, every non-uniform PPT distinguisher $\mathcal{D}$ and every $\lambda \in \mathbb{N}$:*

$$\begin{aligned} \big|\Pr\left[r \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}; (C_0, C_1) := \mathsf{Samp}(1^\lambda, r); \tilde{C} \leftarrow \mathsf{diO}(1^\lambda, C_0) : 1 \leftarrow \mathcal{D}(r, \tilde{C})\right] - \\ \Pr\left[r \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}; (C_0, C_1) := \mathsf{Samp}(1^\lambda, r); \tilde{C} \leftarrow \mathsf{diO}(1^\lambda, C_1) : 1 \leftarrow \mathcal{D}(r, \tilde{C})\right]\big| \\ = \mathsf{negl}(\lambda). \end{aligned} \quad (3)$$

Ishai et al. [16] conjecture that Garg et al.'s [13] $i\mathcal{O}$ construction satisfies their notion of public-coin $di\mathcal{O}$.

2.3 Non-interactive Proof Systems

An efficient non-interactive proof system in the *common-random-string* (CRS) model for a language $L \in \mathsf{NP}$ consists of PPT prover P and verifier V sharing a uniformly random string *crs*. On input a statement and a witness, P outputs a proof; V, on input a statement and a proof outputs 0 or 1. We require proof systems to be complete (honestly generated proofs verify) and sound (no adversary can produce a a valid proof of a false statement).

A non-interactive proof system is *zero-knowledge* if proofs of true statements reveal nothing beyond their validity. This is formalized by requiring the existence of a PPT simulator $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$ that on input a true statement produces a CRS and a proof that are computationally indistinguishable from real ones.

Definition 5 (NIZK). *A tuple of PPT algorithms $\mathsf{NIZK} = (\mathsf{G}, \mathsf{P}, \mathsf{V}, \mathsf{S})$ is a statistically sound non-interactive zero-knowledge (NIZK) proof system in the common-random-string model for $L \in \mathsf{NP}$ with witness relation R if we have:*

1. *Perfect completeness: For every (η, w) such that $R(\eta, w) = 1$, it holds that*

$$\Pr\left[crs \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}; \pi \leftarrow \mathsf{P}(crs, \eta, w) : \mathsf{V}(crs, \eta, \pi) = 1\right] = 1.$$

2. *Statistical soundness:*

$$\Pr\left[crs \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} : \exists\,(\eta, \pi) \ s.t.\ \eta \notin L \ \wedge\ \mathsf{V}(crs, \eta, \pi) = 1\right] = \mathsf{negl}(\lambda). \quad (4)$$

3. *Computational zero-knowledge: For every (η, w) such that $R(\eta, w) = 1$, and non-uniform PPT adversary $\mathcal{A}$, it holds that*

$$\begin{aligned} \big|\Pr\left[crs \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}; \ \pi \leftarrow \mathsf{P}(crs, \eta, w) : \mathcal{A}(crs, \eta, \pi) = 1\right] - \\ \Pr\left[(crs, \tau) \leftarrow \mathsf{S}_1(1^\lambda, \eta); \ \pi \leftarrow \mathsf{S}_2(crs, \tau, \eta) : \mathcal{A}(crs, \eta, \pi) = 1\right]\big| = \mathsf{negl}(\lambda). \end{aligned} \quad (5)$$

A succinct non-interactive argument of knowledge (SNARK) is a computationally sound NI proof-of-knowledge system with universally succinct proofs. A proof for a statement η is *succinct* if its length and verification time are bounded by a fixed polynomial in the statement length $|\eta|$. We define SNARK systems in the common-random-string model following Bitansky et al. [5,6,16].

Definition 6 (The Universal Relation $\mathcal{R}_\mathcal{U}$ [3]). *The universal relation $\mathcal{R}_\mathcal{U}$ is the set of instance-witness pairs of the form $((M,m,t),w)$ where M is a TM accepting an input-witness pair (m,w) within t steps. In particular $|w| \leq t$.*

Definition 7 (SNARK). *A pair of PPT algorithms (P,V), where V is deterministic, is a succinct non-interactive argument of knowledge (SNARK) in the common-random-string model for a language $\mathcal{L}$ with witness relation $\mathcal{R} \subseteq \mathcal{R}_\mathcal{U}$ if there exist polynomials p,ℓ,q* independent *of $\mathcal{R}$ such that the following hold:*

1. *Completeness: For every $(\eta=(M,m,t),w) \in \mathcal{R}$, it holds that*

$$\Pr\left[crs \leftarrow \{0,1\}^{poly(\lambda)};\ \pi \leftarrow \mathsf{P}(crs,\eta,w) :\ \mathsf{V}(crs,\eta,\pi)=1\right]=1.$$

   Moreover, P runs in time $q(\lambda,|\eta|,t)$.
2. *(Adaptive) Soundness: For every PPT adversary $\mathcal{A}$:*

$$\Pr\left[crs \leftarrow \{0,1\}^{poly(\lambda)};\ (\eta,\pi) \leftarrow \mathcal{A}(crs) :\ \eta \notin \mathcal{L} \ \wedge\ \mathsf{V}(crs,\eta,\pi)=1\right]=\mathsf{negl}(\lambda).$$

3. *(Adaptive) Argument of knowledge: For every PPT adversary $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}_\mathcal{A}$ such that*

$$\Pr\left[\begin{array}{l|ll} crs \leftarrow \{0,1\}^{poly(\lambda)};\ r \leftarrow \{0,1\}^{poly(\lambda)} & & (\eta,w)\notin\mathcal{R}\ \wedge \\ (\eta,\pi):=\mathcal{A}(crs;r);\ w \leftarrow \mathcal{E}_\mathcal{A}(1^\lambda,crs,r) & : & \mathsf{V}(crs,\eta,\pi)=1 \end{array}\right]=\mathsf{negl}(\lambda).$$

4. *Succinctness: For all $(crs,\eta,w) \in \{0,1\}^{poly(\lambda)} \times \mathcal{R}$, the length of an honestly generated proof $\pi \leftarrow \mathsf{P}(crs,\eta,w)$ is bounded by $\ell(\lambda,\log t)$ and the running time of $\mathsf{V}(crs,\eta,\pi)$ is bounded by $p(\lambda+|\eta|)=p(\lambda+|M|+|m|+\log t)$.*

Bitansky et al. [5] construct SNARKs for $\mathcal{R}_c \subset \mathcal{R}_\mathcal{U}$ where $t=|m|^c$ and c is a constant, based on knowledge-of-exponent assumptions [6] and extractable collision-resistant hash functions (ECRHF) [5]. These are both non-falsifiable assumptions, but Gentry and Wichs [14] prove that SNARKs cannot be built from falsifiable assumptions via black-box reductions. Relying on exponentially hard one-way functions and ECRHF, [5] construct SNARKs for $\mathcal{R}_\mathcal{U}$.

2.4 Commitment Schemes

A commitment scheme CS for a message space $\mathcal{M} \not\ni \perp$ consists of the following PPT algorithms: On input 1^λ, Setup outputs a commitment key ck; on input ck and a message $m \in \mathcal{M}$, Com outputs a commitment c and an opening d; on input ck, c and d, Open opens c to a message m or $\perp$. Besides correctness (commitments open to the committed message), we require *computational hiding* (no PPT adversary can distinguish commitments to messages of his choice) and *statistical binding* (no unbounded adversary can find some c and two openings d,d', which open c to two different messages, except with negligible probability over the choice of ck). A formal definition is given in the full version [1].

2.5 Collision-Resistant Hash Functions

A family of hash functions is *collision-resistant* (CR) if for a uniformly sampled function H it is hard to find two values that map to the same image under H. It is *public-coin* CR if this is hard even when given the coins used to sample H. A formal definition is given in the full version [1].

2.6 Functional Signatures

Functional signatures were introduced by Boyle et al. [10]. They generalize the concept of digital signatures by letting the holder of a secret key sk derive keys sk_f for functions f.[1] Such a key sk_f enables signing (only) messages in the range of f: running $\mathsf{Sign}(f, sk_f, w)$ produces a signature on $f(w)$.

Definition 8 (Functional Signatures [10]). *A functional signature scheme for a message space $\mathcal{M} \not\ni \bot$ and a function family $\mathcal{F}_\lambda = \{f : \mathcal{D}_f \to \mathcal{R}_f\}_\lambda$ with $\mathcal{R}_f \subseteq \mathcal{M}$ is a tuple of PPT algorithms $\mathsf{FS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ where:*

- $(msk, mvk) \leftarrow \mathsf{Setup}(1^\lambda)$*: On input a security parameter 1^λ,* Setup *outputs a master signing and verification key.*
- $sk_f \leftarrow \mathsf{KeyGen}(msk, f)$*: On input msk and a function $f \in \mathcal{F}_\lambda$,* KeyGen *outputs a signing key sk_f.*
- $(f(w), \sigma) \leftarrow \mathsf{Sign}(f, sk_f, w)$*: On input $f \in \mathcal{F}_\lambda$, a signing key sk_f for f, and $w \in \mathcal{D}_f$,* Sign *outputs a signature on $f(w) \in \mathcal{M}$.*
- $b = \mathsf{Verify}(mvk, m, \sigma)$*: On input a master verification key mvk, a message $m \in \mathcal{M}$, and signature σ,* Verify *outputs $b \in \{0, 1\}$.*

A functional signature is *correct* if correctly generated signatures verify, and is secure if it satisfies *unforgeability*, *function privacy*, and *succinctness*.

Unforgeability states that even given oracles that generate signatures and functional signing keys, it must be hard to produce a valid signature on a message that was not submitted to the signing oracle and that cannot be signed using a key obtained from the key oracle. *Function privacy* states that signatures neither reveal the function associated to the secret key nor the used preimage w. *Succinctness* requires that the size of a signature is independent of $|w|$ and $|f|$. A formal security definition is given in the full version [1].

Boyle et al. [10] construct functional signatures based on zero-knowledge SNARKs.

3 Functional Signatures with Obliviously Samplable Keys

We introduce and construct a new primitive we call *functional signatures with obliviously samplable keys* (FSwOSK), which will be central to achieving short

[1] In [10], f is given as a circuit, but in their construction of functional encryption, Boyle et al. [9] allow f to be a Turing machine. In this work we adopt the latter definition.

$\underline{\mathbf{Exp}^{\text{ind-}b}_{\mathsf{S},\mathcal{A}}(\lambda)}$

$(st, m) \leftarrow \mathcal{A}_1(1^\lambda)$
If $b = 0$
  $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$; $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$
Else $(\mathsf{vk}, \sigma) \leftarrow \mathsf{OSmp}(1^\lambda, m)$
Return $b' \leftarrow \mathcal{A}_2(st, \mathsf{vk}, \sigma)$

Fig. 2. The oblivious-indist. game

$\underline{\mathbf{Exp}^{\text{obl-uf}}_{\mathsf{S},\mathcal{A}}(\lambda)}$

$(st, m) \leftarrow \mathcal{A}_1(1^\lambda)$
$r \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$; $(\mathsf{vk}, \sigma) \leftarrow \mathsf{OSmp}(1^\lambda, m; r)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}_2(st, r)$
Return 1 iff $m^* \neq m$
  $\wedge\, \mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$

Fig. 3. The oblivious-unforgeability game

keys for CPRF with unbounded inputs. We first extend a (standard) signature scheme by an extra functionality that given a message m allows one to sample a verification key together with a signature on m in an oblivious way. This means that, while the key and the signature look like regularly generated ones, it is hard to forge a signature on a different message under this key, even when given the coins used to sample the key/signature pair. We call this primitive *signatures with obliviously samplable signatures* (SwOSS) and construct it from one-way functions and NIZK by adapting a signature scheme due to Bellare and Goldwasser [4]. We then combine this scheme with SNARKs in order to construct our FSwOSK following the construction of a (standard) functional signature scheme of Boyle et al. [10].

3.1 Signature Schemes with Obliviously Samplable Signatures

Definition 9 (SwOSS). *Let* $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a (standard) signature scheme that is existentially unforgeable under chosen-message attacks (EUF-CMA) with message space* $\mathcal{M} \not\ni \perp$*. We say* S *has obliviously samplable signatures if there exists a PPT algorithm* OSmp *such that:*

- $(\mathsf{vk}, \sigma) \leftarrow \mathsf{OSmp}(1^\lambda, m)$*: On input security parameter* 1^λ *and a message* $m \in \mathcal{M}$*,* OSmp *outputs a verification key* vk *and a signature* σ *on* m*.*

SwOSS S *is secure if it satisfies (with experiments defined in Figs. 2 and 3):*

1. *Indistinguishability: For every PPT algorithm* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *in* $\mathbf{Exp}^{ind\text{-}b}_{\mathsf{S},\mathcal{A}}(\lambda)$

$$\left|\Pr\left[\mathbf{Exp}^{ind\text{-}0}_{\mathsf{S},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathbf{Exp}^{ind\text{-}1}_{\mathsf{S},\mathcal{A}}(\lambda) = 1\right]\right| = \mathit{negl}(\lambda). \tag{6}$$

2. *Oblivious unforgeability: For every PPT* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *in* $\mathbf{Exp}^{obl\text{-}uf}_{\mathsf{S},\mathcal{A}}(\lambda)$

$$\Pr\left[\mathbf{Exp}^{obl\text{-}uf}_{\mathsf{S},\mathcal{A}}(\lambda) = 1\right] = \mathit{negl}(\lambda). \tag{7}$$

Construction 1 (SwOSS). *Let* $\mathcal{F}_\lambda = \{F : \mathcal{K} \times \{0,1\}^n \to \mathcal{Y}\}$ *be a family of PRFs,* $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ *a perfectly binding commitment scheme for message space* $\mathcal{M}$*, and* $\mathsf{NIZK} = (\mathsf{G}, \mathsf{P}, \mathsf{V}, \mathsf{S}$ *a statistically sound NIZK scheme for*

$$L_\eta := \left\{ (ck, c_0, c_1, y, m) \;\middle|\; \exists\, (k, r) : \begin{matrix} (c_0 = \mathsf{CS.Com}_1(ck, k; r) \wedge y = F(k, m)) \\ \vee\; c_1 = \mathsf{CS.Com}_1(ck, m; r) \end{matrix} \right\} \tag{8}$$

(where Com_1 *denotes the first output of* Com*). Let* $\top \in \mathcal{M}$ *be such that* $\top \notin \mathcal{K}$ *and* $\top \notin \{0,1\}^n$*. Our signatures-with-obliviously-samplable-signatures scheme* $\mathsf{OS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{OSmp})$ *is defined as follows:*

$\underline{(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda) :}$ *On input a security parameter* 1^λ*, compute*

- $k \leftarrow \mathsf{F.Smp}(1^\lambda)$; $crs \leftarrow \{0,1\}^{poly(\lambda)}$; $ck \leftarrow \mathsf{CS.Setup}(1^\lambda)$;
- $(c_0, d_0) := \mathsf{CS.Com}(ck, k)$; $(c_1, d_1) := \mathsf{CS.Com}(ck, \top)$;

return $\mathsf{sk} := (k, r_0), \mathsf{vk} := (crs, ck, c_0, c_1)$

$\underline{\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) :}$ *On input* $\mathsf{sk} = (k, r_0)$ *and* $m \in \mathcal{M}$ *compute*

- $y := F(k, m)$;
- $\pi \leftarrow \mathsf{NIZK.P}(crs, \eta := (ck, c_0, c_1, y, m), (k, r_0))$, *where* $\eta \in L_\eta$ *from* (8);

return $\sigma := (y, \pi)$.

$\underline{b := \mathsf{Verify}(\mathsf{vk}, m, \sigma) :}$ *On input* $\mathsf{vk} = (crs, ck, c_0, c_1)$, *m and* $\sigma = (y, \pi)$,

return $b := \mathsf{NIZK.V}(crs, \eta = (ck, c_0, c_1, y, m), \pi)$.

$\underline{(\mathsf{vk}, \sigma) \leftarrow \mathsf{OSmp}(1^\lambda, m) :}$ *On input* 1^λ *and* $m \in \mathcal{M}$ *, compute*

- $r := r_0 \| r_1 \| r_y \| r_{\mathsf{Setup}} \| crs \| r_{\mathsf{P}} \leftarrow \{0,1\}^{poly(\lambda)}$,
- $y \leftarrow_{r_y} \mathcal{Y}$ // r_y *is used to sample* y *from* $\mathcal{Y}$,
- $ck := \mathsf{CS.Setup}(1^\lambda; r_{\mathsf{Setup}})$,
- $(c_0, d_0) := \mathsf{CS.Com}(ck, \top; r_0)$; $(c_1, d_1) := \mathsf{CS.Com}(ck, m; r_1)$,
- $\pi := \mathsf{NIZK.P}(crs, \eta := (ck, c_0, c_1, y, m), w := (m, r_1); r_{\mathsf{P}})$;

return $\mathsf{vk} := (crs, ck, c_0, c_1)$ *and* $\sigma := (y, \pi)$.

Theorem 1. *Scheme* OS *in Construction 1 is an EUF-CMA-secure signature scheme with obliviously samplable signatures.*

Proof. We need to show that $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is (standard) EUF-CMA-secure and prove indistinguishability (6) and oblivious unforgeability (7). The proof of EUF-CMA is analogous to that of Bellare and Goldwasser's [4] (noting that the second clause in (8) is always false) and is therefore omitted.

Indistinguishability: Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT adversary that non-negligibly distinguishes honestly generated ($\mathbf{Exp}^{\text{ind-0}}_{\mathsf{OS},\mathcal{A}}(\lambda)$) and obliviously sampled verification key-signature pairs ($\mathbf{Exp}^{\text{ind-1}}_{\mathsf{OS},\mathcal{A}}(\lambda)$). Our proof will be by game hopping and we define a series of games $\mathbf{Exp}^{(0)} := \mathbf{Exp}^{\text{ind-0}}_{\mathsf{OS},\mathcal{A}}(\lambda)$, $\mathbf{Exp}^{(1)}, \ldots, \mathbf{Exp}^{(5)} := \mathbf{Exp}^{\text{ind-1}}_{\mathsf{OS},\mathcal{A}}(\lambda)$ and show that for $c = 0, \ldots, 4$, $\mathbf{Exp}^{(c)}$ and $\mathbf{Exp}^{(c+1)}$ are computationally indistinguishable. In $\mathbf{Exp}^{(0)}$ the adversary obtains *vk* output by KeyGen and σ output by Sign as defined in Construction 1.

$\mathbf{Exp}^{\text{ind-}b}_{\mathsf{FS},\mathcal{A}}(\lambda)$

$(st, f) \leftarrow \mathcal{A}_1(1^\lambda)$
If $b = 0$
  $(msk, mvk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $sk_f \leftarrow \mathsf{KeyGen}(msk, f)$
Else $(mvk, sk_f) \leftarrow \mathsf{OSmp}(1^\lambda, f)$
Return $b' \leftarrow \mathcal{A}_2(st, mvk, sk_f)$

Fig. 4. The oblivious-indist. game

$\mathbf{Exp}^{\text{obl-uf}}_{\mathsf{FS},\mathcal{A}}(\lambda)$

$(st, f) \leftarrow \mathcal{A}_1(1^\lambda)$
$r \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$
$(mvk, sk_f) \leftarrow \mathsf{OSmp}(1^\lambda, f; r)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}_2(st, r)$
Return 1 iff $m^* \notin \mathcal{R}_f$
  $\wedge\, \mathsf{Verify}(mvk, m^*, \sigma^*) = 1$

Fig. 5. The oblivious-unforgeability game

$\mathbf{Exp}^{(1)}$ differs from $\mathbf{Exp}^{(0)}$ in that the CRS for the NIZK and the proof π are simulated. By zero knowledge of NIZK the game is indistinguishable from $\mathbf{Exp}^{(0)}$.
$\mathbf{Exp}^{(2)}$ differs from $\mathbf{Exp}^{(1)}$ in that c_0 commits to $\top$ rather than a PRF key k. By computational hiding of CS, this is indistinguishable for PPT adversaries (note that r_0 is not used elsewhere in the game).
$\mathbf{Exp}^{(3)}$ differs from $\mathbf{Exp}^{(2)}$ in that c_1 commits to m rather than $\top$. Again, by hiding of CS (and since r_1 is not used anywhere), this is indistinguishable.
$\mathbf{Exp}^{(4)}$ differs from $\mathbf{Exp}^{(3)}$ in that $y \leftarrow \mathcal{Y}$ is random rather than $y := \mathsf{F}(k, m)$. Pseudorandomness of F guarantees this change is indistinguishable to PPT adversaries (note that k is not used anywhere else in the game).
$\mathbf{Exp}^{(5)}$ differs from $\mathbf{Exp}^{(4)}$ in that the CRS *crs* for the NIZK is chosen at random (rather than simulated) and π is computed by $\mathsf{NIZK.P}$. Again, this is indistinguishable by zero knowledge of NIZK.

Oblivious unforgeability. This follows from soundness of NIZK and the binding property of CS. OSmp sets c_0 to a commitment of $\top$ and c_1 to a commitment of m. If $\mathcal{A}$ manages to output a signature (y^*, π^*) that is valid on message $m^* \neq m$, i.e., $\mathsf{NIZK.V}(crs, (ck, c_0, c_1, y^*, m^*), \pi^*) = 1$, then by soundness of NIZK, $(ck, c_0, c_1, y^*, m^*) \in L_\eta$ (8), meaning that either c_0 is a commitment to a valid PRF key or c_1 is a commitment to m^*. Either case would contradict the binding property of the commitment scheme.

This proves Theorem 1. A formal proof is given in the full version [1].

3.2 Functional Signature Schemes with Obliviously Samplable Keys

Definition 10. (FSwOSK). *Let* $\mathsf{FS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a functional signature scheme (Definition 8).* FS has obliviously samplable keys *if there exists a PPT algorithm:*

$(mvk, sk_f) \leftarrow \mathsf{OSmp}(1^\lambda, f)$*: On input* 1^λ *and a function* $f \in \mathcal{F}_\lambda$, OSmp *outputs a master verification key* mvk *and a functional signing key* sk_f *for* f*.*

FSwOSK FS *is secure if it is a secure functional signature scheme that additionally satisfies the following:*

1. *Indistinguishability: For every PPT* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *in* $\mathbf{Exp}^{ind\text{-}b}_{\mathsf{FS},\mathcal{A}}(\lambda)$ *(Fig. 4):*

$$|\Pr\left[\mathbf{Exp}^{ind\text{-}0}_{\mathsf{FS},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathbf{Exp}^{ind\text{-}1}_{\mathsf{FS},\mathcal{A}}(\lambda) = 1\right]| = \mathit{negl}(\lambda).$$

2. *Oblivious unforgeability: For every PPT* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *in* $\mathbf{Exp}^{obl\text{-}uf}_{\mathsf{FS},\mathcal{A}}(\lambda)$ *(Fig. 5):*

$$\Pr\left[\mathbf{Exp}^{obl\text{-}uf}_{\mathsf{FS},\mathcal{A}}(\lambda) = 1\right] = \mathit{negl}(\lambda).$$

We next show that if in the construction of functional signatures of Boyle et al. [10] we replace the signature scheme by a SwOSS (Definition 9) then we obtain a FSwOSK. As a first step Boyle et al. [10, Theorem 3.3] construct $\mathsf{FS}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$, which does not satisfy function privacy nor succinctness, but which is unforgeable if the underlying signature scheme is EUF-CMA. Relying on adaptive zero-knowledge SNARKs for NP, they then transform FS' into a secure FS scheme [10, Theorem 3.4].

We first enhance their scheme FS' by an oblivious sampler OSmp' so it also satisfies indistinguishability and oblivious unforgeability, as defined in Definition 10.

Construction 2. *Let* $\mathsf{OS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{OSmp})$ *be a secure SwOSS and* SS *an EUF-CMA-secure signature scheme. For a message space* $\mathcal{M} \not\ni \bot$ *and a function family* $\mathcal{F}_\lambda = \{f : \mathcal{D}_f \to \mathcal{R}_f \subseteq \mathcal{M}\}_\lambda$*, we construct* FS' *as follows:*

$(msk, mvk) \leftarrow \mathsf{FS}'.\mathsf{Setup}(1^\lambda)$: *Return* $(msk, mvk) \leftarrow \mathsf{OS.KeyGen}(1^\lambda)$.

$sk_f \leftarrow \mathsf{FS}'.\mathsf{KeyGen}(msk, f)$: *On input* msk *and* $f \in \mathcal{F}_\lambda$*, compute* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SS.KeyGen}(1^\lambda)$, $\sigma_{f\|\mathsf{vk}} \leftarrow \mathsf{OS.Sign}(msk, f\|\mathsf{vk})$ *; return* $sk_f := (f\|\mathsf{vk}, \sigma_{f\|\mathsf{vk}}, \mathsf{sk})$.

$(f(w), \sigma) \leftarrow \mathsf{FS}'.\mathsf{Sign}(f, sk_f, w)$: *On input* $f \in \mathcal{F}_\lambda$*, key* $sk_f = (f\|\mathsf{vk}, \sigma_{f\|\mathsf{vk}}, sk)$ *for* f *and* $w \in \mathcal{D}_f$ *, compute* $\sigma_w \leftarrow \mathsf{SS.Sign}(\mathsf{sk}, w)$*; return* $\sigma := (f\|\mathsf{vk}, \sigma_{f\|\mathsf{vk}}, w, \sigma_w)$.

$b = \mathsf{FS}'.\mathsf{Verify}(mvk, m, \sigma)$: *Given* mvk*,* $m \in \{0,1\}^*$*,* $\sigma = (f\|\mathsf{vk}, \sigma_{f\|\mathsf{vk}}, w, \sigma_w)$*; return* $\mathsf{OS.Verify}(mvk, f\|\mathsf{vk}, \sigma_{f\|\mathsf{vk}}) = 1 = \mathsf{SS.Verify}(\mathsf{vk}, w, \sigma_w) \wedge m = f(w)$.

$(mvk, sk_f) \leftarrow \mathsf{FS}'.\mathsf{OSmp}(1^\lambda, f)$: *Given* 1^λ *and* $f \in \mathcal{F}_\lambda$*, pick* $r_\mathsf{G}, r_\mathsf{O} \leftarrow \{0,1\}^{poly(\lambda)}$*, set* $(\mathsf{sk}, \mathsf{vk}) := \mathsf{SS.KeyGen}(1^\lambda; r_\mathsf{G})$, $(mvk, \sigma_{f\|\mathsf{vk}}) := \mathsf{OS.OSmp}(1^\lambda, f\|\mathsf{vk}; r_\mathsf{O})$*; return* mvk *and* $sk_f := (f\|\mathsf{vk}, \sigma_{f\|\mathsf{vk}}, \mathsf{sk})$.

Theorem 2. FS' *of Construction 2 is a FSwOSK that satisfies correctness, unforgeability, indistinguishability and oblivious unforgeability (but neither function privacy nor succinctness).*

Theorem 2 is formally proved in the full version [1] and we give some proof intuition here. Theorem 3.3 in [10] proves that $(\mathsf{FS}'.\mathsf{Setup}, \mathsf{FS}'.\mathsf{KeyGen}, \mathsf{FS}'.\mathsf{Sign}, \mathsf{FS}'.\mathsf{Verify})$ is a functional signature scheme that is correct and unforgeable. What remains then is to show that $\mathsf{FS.OSmp}'$ satisfies both indistinguishability (Item 1. in Definition 10) and oblivious unforgeability (Item 2.).

Note that a FSwOSK master verification key is a SwOSS verification key, and a FswOSK functional signing key is a SwOSS signature; thus an obliviously samplable pair for FSwOSK translates to a pair for SwOSS; indistinguishability

for FSwOSK reduces thus to indistinguishability for SwOSS. Similarly, oblivious unforgeability for FSwOSK reduces to oblivious unforgeability of SwOSS (note that in this game the adversary cannot ask for functional signatures, so EUF-CMA of the regular signature scheme is not needed).

Next we show that the transformation of [10] applies to our scheme FS', and therefore the transformed FS is a FSwOSK satisfying Definition 10.

Theorem 3. *Assuming an adaptive zero-knowledge SNARK system for* NP, FS' *from Construction 2 can be transformed into a secure FSwOSK scheme* FS.

Proof (Proof sketch). The construction and proof of the theorem are exactly the same as those of Theorem 3.4 of [10], and therefore we only give an intuitive argument and refer the reader to [10] for more details.

First observe that in FS' a signature $\sigma := (f\|\mathsf{vk}, \sigma_{f\|\mathsf{vk}}, w, \sigma_w)$ on $f(w)$ contains both f and w in the clear and is therefore neither function-private nor succinct. In the new scheme FS a signature on m is instead a zero-knowledge SNARK proof π of knowledge of the following: f, vk, a signature $\sigma_{f\|vk}$ on $f\|vk$ that verifies under mvk, an element w such that $f(w) = m$, and a signature σ on w, valid under vk. Now function privacy reduces to zero knowledge and succinctness of signatures reduces to succinctness of the underlying SNARK.

4 Constrained PRFs for Unbounded Inputs

In this section we construct a family of constrained PRFs for unbounded inputs such that a constrained key is simply a (functional) signature on the constraining TM M. As a warm-up, we review the construction of [2] where a constrained key is a $di\mathcal{O}$ obfuscation of a circuit that depends on the size of the constraining TM M. In particular, the circuit verifies a SNARK for the following relation.

Definition 11 (R_{legit})**.** *We define the relation* $R_{legit} \subset \mathcal{R}_{\mathcal{U}}$ *(with* $\mathcal{R}_{\mathcal{U}}$ *from Definition 6) to be the set of instance-witness pairs* $(((H, M), h, t), x)$ *such that* M *and* H *are descriptions of a TM and a hash function,* $M(x) = 1$ *and* $H(x) = h$ *within* t *steps. We let* L_{legit} *be the language corresponding to* R_{legit}*. For notational convenience, abusing notation, we write* $((H, M, h), x) \in R_{legit}$ *to mean* $(((H, M), h, t), x) \in R_{legit}$ *while implicitly setting* $t = 2^\lambda$.

Remark 2. Let $t = 2^\lambda$ in the definition of R_{legit}; then by succinctness of SNARKs (Definition 7), the length of a SNARK proof is bounded by $\ell(\lambda)$ and its verification time is bounded by $p(\lambda + |M| + |H| + |h|)$, where p, ℓ are a priori fixed polynomials that do not depend on R_{legit}.

Construction 3 [2]. *Let* $\mathcal{PF}_\lambda = \{\mathsf{PF}\colon \mathcal{K} \times \{0,1\}^n \to \mathcal{Y}\}$ *be a selectively secure puncturable PRF,* $\mathcal{H}_\lambda = \{H\colon \{0,1\}^* \to \{0,1\}^n\}_\lambda$ *a family of public-coin CR hash functions,* diO *a public-coin* $di\mathcal{O}$ *obfuscator for a family of polynomial-size circuits* $\mathcal{P}_\lambda$*, and* SNARK *a SNARK system for* R_{legit} *(Definition 11). A family of selectively secure PRFs* $\mathcal{F}_\lambda = \{F\colon \mathcal{K} \times \{0,1\}^* \to \mathcal{Y}\}$ *constrained w.r.t. to any polynomial-size family of TMs* $\mathcal{M}_\lambda$ *is defined as follows:*

$K \leftarrow \mathsf{F.Smp}(1^\lambda)$: *Sample* $k \leftarrow \mathsf{PF.Smp}(1^\lambda)$, $H \leftarrow \mathsf{H.Smp}(1^\lambda)$, $crs \leftarrow \{0,1\}^{poly(\lambda)}$; *return* $K := (k, H, crs)$.

$k_M \leftarrow \mathsf{F.Constr}(K, M)$: *On input* $K = (k, H, crs)$ *and* $M \in \mathcal{M}_\lambda$, *define*

$$P_{M,H,crs,k}(h,\pi) := \begin{cases} \mathsf{PF.Eval}(k,h) & \text{if } \mathsf{SNARK.V}(crs,(H,M,h),\pi) = 1 \\ \bot & \text{otherwise} \end{cases} \tag{9}$$

compute $\widetilde{P} \leftarrow \mathsf{diO}(1^\lambda, P_{M,H,crs,k})$ *and output* $k_M := (M, \widetilde{P}, H, crs)$.

$y := \mathsf{F.Eval}(\kappa, x)$: *On input* $\kappa \in \mathcal{K} \cup \mathcal{K}_\mathcal{M}$ *and* $x \in \{0,1\}^*$, *do the following:*

- *If* $\kappa \in \mathcal{K}$, $\kappa = (k, H, crs)$: *return* $\mathsf{PF.Eval}(k, H(x))$.
- *If* $\kappa = (M, \widetilde{P}, H, crs) \in \mathcal{K}_\mathcal{M}$: *if* $M(x) = 1$, *let* $h := H(x)$ *(thus* $(H, M, h) \in L_{legit}$*),* $\pi \leftarrow \mathsf{SNARK.P}(crs, (H, M, h), x)$ *and return* $y := \widetilde{P}(h, \pi)$.

The drawback of Construction 3 is that a constrained key for a TM M is a $di\mathcal{O}$-obfuscated circuit and is therefore large. In our construction below we use FSwOSK to define a constrained key k_M simply as a functional signature on M. As in Construction 3, our constrained PRF F is defined as $\mathsf{F}(k,x) = \mathsf{PF}(k, H(x))$, where PF is a puncturable PRF and H is a collision-resistant hash function. To enable evaluating F given a constrained key k_M, in the setup we output as a public parameter a $di\mathcal{O}$-obfuscation of a circuit P (defined in (10) below) that on input (M, h, π, σ) outputs $\mathsf{PF}(k,h)$ which is equal to $\mathsf{F}(k,x)$ if π is a valid SNARK proving knowledge of some x such that $M(x) = 1$ and $h = H(x)$, and moreover σ is a valid functional signature on M; and outputs $\bot$ otherwise.

Construction 4 (TM CPRF with short keys). *Let* $\mathcal{PF}_\lambda = \{\mathsf{PF}\colon \mathcal{K} \times \{0,1\}^n \to \mathcal{Y}\}$ *be a selectively secure puncturable PRF ,* $\mathcal{H}_\lambda = \{H\colon \{0,1\}^* \to \{0,1\}^n\}_\lambda$ *a family of public-coin collision-resistant hash functions,* $\mathsf{FS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{OSmp})$ *a FSwOSK scheme, diO a public-coin differing-input obfuscator for a family of poly-size circuits* $\mathcal{P}_\lambda$*, and SNARK a SNARK system in the common-random-string model for* R_{legit} *(cf. Definition 11).*

We construct a family of PRFs $\mathcal{F}_\lambda = \{\mathsf{F}\colon \mathcal{K} \times \{0,1\}^* \to \mathcal{Y}\}$ *constrained w.r.t. to a polynomial-size family of Turing machines* $\mathcal{M}_\lambda$ *as follows:*

$K \leftarrow \mathsf{F.Smp}(1^\lambda)$:

- $H \leftarrow \mathsf{H.Smp}(1^\lambda)$.
- $crs \leftarrow \{0,1\}^{poly(\lambda)}$.
- $(msk, mvk) \leftarrow \mathsf{FS.Setup}(1^\lambda)$.
- $sk_{f_I} \leftarrow \mathsf{FS.KeyGen}(msk, f_I)$ *where* $f_I(M) := M$.
- $k \leftarrow \mathsf{PF.Smp}(1^\lambda)$.
- $\widetilde{P} \leftarrow \mathsf{diO}(1^\lambda, P)$ *where* $P = P_{H,crs,mvk,k} \in \mathcal{P}_\lambda$ *is defined as:*

$$P(M,h,\pi,\sigma) := \begin{cases} \mathsf{PF.Eval}(k,h) & \text{if } \mathsf{SNARK.V}\big(crs,(H,M,h),\pi\big) = 1 \\ & \quad \wedge\ \mathsf{FS.Verify}(mvk, M, \sigma) = 1 \\ \bot & \text{otherwise} \end{cases} \tag{10}$$

– *Set* $pp = (H, crs, mvk, \widetilde{P})$ *and return* $K := (k, sk_{f_I}, pp)$.

$k_M \leftarrow \mathsf{F.Constr}(K, M)$: *On input* $K = (k, sk_{f_I}, pp)$ *and* $M \in \mathcal{M}_\lambda$, *compute* $(M, \sigma) \leftarrow \mathsf{FS.Sign}(I, sk_{f_I}, M)$ *and return* $k_M := (M, \sigma, pp)$.

$y := \mathsf{F.Eval}(\kappa, x)$: *On input* $\kappa \in \mathcal{K} \cup \mathcal{K}_\mathcal{M}$ *and* $x \in \{0,1\}^*$:

– *If* $\kappa \in \mathcal{K}$, $\kappa = (k, sk_{f_I}, pp = (H, crs, mvk, \widetilde{P}))$: *set* $y := \mathsf{PF.Eval}(k, H(x))$.
– *If* $\kappa \in \mathcal{K}_\mathcal{M}$, $\kappa = (M, \sigma, (H, crs, mvk, \widetilde{P}))$: *if* $M(x) = 1$, *set* $h := H(x)$ *(thus* $(H, M, h) \in L_{legit}$*), compute* $\pi \leftarrow \mathsf{SNARK.P}(crs, (H, M, h), x)$, *and return* $y := \widetilde{P}(M, h, \pi, \sigma)$.

Remark 3. The public parameters pp are computed once and for all. As the model for CPRFs defines no public parameters, we formally include them in k_M. Note that $\mathcal{P}_\lambda$ is a circuit family with input length $|M| + n + |\pi| + |\sigma|$ where $|\pi|$ is upper bounded by $\ell(\lambda)$ even for an exponentially long x (cf. Remark 2).

Let us now argue why we need functional signatures with obliviously samplable keys in order to prove our construction secure.

If we could replace the PRF key k by a punctured one $k^* := k_{H(x^*)}$ then $\mathsf{F}(k, x^*)$ would look random, as required for selective security of F. The obfuscated circuit P would thus use k^* instead of k. But obfuscations of P_k and P_{k^*} are only indistinguishable if it is hard to find an input on which they differ. And, since we use public-coin diO, this should be even hard when given all coins used to produce P_k and P_{k^*}.

In the security experiment the adversary can query keys for machines M with $M(x^*) = 0$ and when fed to P_k and P_{k^*}, both output the same. However, if the adversary manages to forge a signature on some $\hat{M}$ with $\hat{M}(x^*) = 1$ then P_k outputs $\mathsf{F}(k, x^*)$, but P_{k^*}, using a punctured key, outputs $\perp$.

The tricky part is to break some unforgeability notion when this happens. The differing-input sampler that computes P_k and P_{k^*} must simulate the experiment for $\mathcal{A}$ and thus create signatures to answer key queries. This is why we need functional signatures, as then the sampler can use a signing key sk_{f^*}, which only allows signing of machines with $M(x^*) = 0$, to answer key queries. FS unforgeability guarantees that even given such a key it is hard to compute a signature on some $\hat{M}$ with $\hat{M}(x^*) = 1$.

The next problem is that finding a differing input (and thus a forgery on $\hat{M}$) should be hard *even when given all coins*, so in particular the coins to create the signature verification key mvk contained in P_k and P_{k^*}; thus it would be easy to "forge a signature". This is why we need FSwOSK, as they allow to sample a verification key together with sk_{f^*} and even given the coins, forgeries should be hard.

Theorem 4. $\mathcal{F}_\lambda$ *of Construction 4 is a selectively secure family of constrained PRFs with input space* $\{0,1\}^*$ *for which constrained keys can be derived for any set that can be decided by a polynomial-size Turing machine.*

$\underline{\mathbf{Exp}^{b,(c)}_{\mathcal{F},\mathcal{A}}(\lambda)}$ // $c \in \{0,1,2,3,4\}$

$(x^*, st) \leftarrow \mathcal{A}_1(1^\lambda)$
$H \leftarrow \mathsf{H.Smp}(1^\lambda)$; $crs \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$
$\underline{\text{If } c \leq 1}$
  $(msk, mvk) \leftarrow \mathsf{FS.Setup}(1^\lambda)$
  Define f_I and f_{x^*} as in (11) and pad them to the same length.
  $sk_{f_I} \leftarrow \mathsf{FS.KeyGen}(msk, f_I)$
  $sk_{f_{x^*}} \leftarrow \mathsf{FS.KeyGen}(msk, f_{x^*})$
Else
  $(mvk, sk_{f_{x^*}}) \leftarrow \mathsf{FS.OSmp}(1^\lambda, f_{x^*})$
$k \leftarrow \mathsf{PF.Smp}(1^\lambda)$
$k_{h^*} \leftarrow \mathsf{PF.Constr}(k, \{0,1\}^n \setminus \{H(x^*)\})$
$\underline{\text{If } c \leq 2 \text{ then}}$
  $P := P_{H,crs,mvk,k}$ as defined in (10)
Else
  $P := P_{H,crs,mvk,k_{h^*}}$ as defined in (10)
$\widetilde{P} \leftarrow \mathsf{diO}(1^\lambda, P)$
$pp := (H, crs, mvk, \widetilde{P})$
If $b = 1$, $y^* := \mathsf{PF.Eval}(k, H(x^*))$, else $y^* \leftarrow \mathcal{Y}$
$b' \leftarrow \mathcal{A}_2^{\textsc{Constr}(\cdot),\textsc{Eval}(\cdot)}(st, y^*)$; return b'

Oracle $\textsc{Constr}(M)$

If $M \notin \mathcal{M}_\lambda \vee M(x^*) = 1$
  Return $\perp$
$\underline{\text{If } c = 0}$
  $(M, \sigma) \leftarrow \mathsf{FS.Sign}(f_I, sk_{f_I}, M)$
Else
  $(M, \sigma) \leftarrow \mathsf{FS.Sign}(f_{x^*}, sk_{f_{x^*}}, M)$
Return $k_M := (M, \sigma, pp)$

Oracle $\textsc{Eval}(x)$

If $x = x^*$
  Return $\perp$
$\underline{\text{If } c \leq 3}$
  $y := \mathsf{PF.Eval}(k, H(x))$
Else
  If $H(x) = H(x^*)$, abort
  Else $y := \mathsf{PF.Eval}(k_{h^*}, H(x))$
Return y

Fig. 6. Hybrids used in the proof of Theorem 4

Proof. Let $\mathcal{A}$ be a PPT adversary for the game $\mathbf{Exp}^{(\emptyset,\{\textsc{Constr},\textsc{Eval}\}),b}_{\mathcal{F},\mathcal{A}}(\lambda)$, as defined in Fig. 6, which we abbreviate as $\mathbf{Exp}^b$. We need to show that $\mathbf{Exp}^0$ and $\mathbf{Exp}^1$ are indistinguishable. Our proof will be by game hopping and we define a series of hybrid games $\mathbf{Exp}^{b,(0)} := \mathbf{Exp}^b$, $\mathbf{Exp}^{b,(1)}$,$\mathbf{Exp}^{b,(2)}$, $\mathbf{Exp}^{b,(3)}$, $\mathbf{Exp}^{b,(4)}$ and show that for $b = 0, 1$ and $c = 0, 1, 2, 3$ the games $\mathbf{Exp}^{b,(c)}$ and $\mathbf{Exp}^{b,(c+1)}$ are indistinguishable. Finally we show that $\mathbf{Exp}^{0,(4)}$ and $\mathbf{Exp}^{1,(4)}$ are also indistinguishable, which concludes the proof. All games are defined in Fig. 6, using the following definitions:

$$f_I \colon M \mapsto M, \qquad f_{x^*} \colon M \mapsto \begin{cases} M & \text{if } M(x^*) = 0 \\ \perp & \text{otherwise} \end{cases} \tag{11}$$

$\mathbf{Exp}^{b,(0)}$ is the original game $\mathbf{Exp}^{b,(\emptyset,\{\textsc{Constr},\textsc{Eval}\})}_{\mathcal{F},\mathcal{A}}(\lambda)$ for Construction 4. (Note that we padded f_I but, by succinctness, functional signatures (returned by $\textsc{Constr}$) are independent of the length of f.)

$\mathbf{Exp}^{b,(1)}$ differs from $\mathbf{Exp}^{b,(0)}$ by replacing the signing key sk_{f_I} with $sk_{f_{x^*}}$, which only allows to sign machines M with $M(x^*) = 0$.

$\mathbf{Exp}^{b,(2)}$ differs from $\mathbf{Exp}^{b,(1)}$ by replacing the verification/signing key pair $(mvk, sk_{f_{x^*}})$ with an obliviously sampled one.

$\mathbf{Exp}^{b,(3)}$ differs from $\mathbf{Exp}^{b,(2)}$ by replacing the full key of the puncturable PRF PF with one that is punctured at $H(x^*)$ in the definition of P.

$\mathbf{Exp}^{b,(4)}$ differs from $\mathbf{Exp}^{b,(3)}$ by answering EVAL queries using the punctured key k_{h^*} and aborting whenever the adversary queries EVAL on a value that collides with x^* under H.

Intuitively, $\mathbf{Exp}^{b,(0)}(\lambda)$ and $\mathbf{Exp}^{b,(1)}(\lambda)$ are computationally indistinguishable as the only difference between them is the use of the signing key sk_{f_I} and $sk_{f_{x^*}}$, respectively, in answering constraining queries. The CONSTR oracle only computes signatures on TMs M with $M(x^*) = 0$. Therefore, f_{x^*} coincides with f_I on all such legitimate queries. By function privacy of FS, signatures generated with f_{x^*} and f_I are computationally indistinguishable.

Proposition 1. $\mathbf{Exp}^{b,(0)}$ *and* $\mathbf{Exp}^{b,(1)}$ *are computationally indistinguishable for* $b = 0, 1$ *if* FS *is a functional signature scheme satisfying function privacy and succinctness.*

The only difference between $\mathbf{Exp}^{b,(1)}$ and $\mathbf{Exp}^{b,(2)}$ is in how mvk and $sk_{f_{x^*}}$ are computed. In $\mathbf{Exp}^{b,(1)}$ the keys mvk (used to define P) and $sk_{f_{x^*}}$ (used to answer CONSTR queries) are generated by $\mathsf{FS.Setup}$ and $\mathsf{FS.KeyGen}$, resp., whereas in $\mathbf{Exp}^{b,(2)}$ they are obliviously sampled together. Indistinguishability of honestly generated and obliviously sampled pairs (Definition 10) of verification/signing key pairs guarantees that this change is indistinguishable to PPT adversaries.

Proposition 2. $\mathbf{Exp}^{b,(1)}$ *and* $\mathbf{Exp}^{b,(2)}$ *are computationally indistinguishable for* $b = 0, 1$ *if* FS *is a FS scheme with obliviously samplable keys.*

It is in the next step that we use the full power of our new primitive FSwOSK. The only difference between $\mathbf{Exp}^{b,(2)}$ and $\mathbf{Exp}^{b,(3)}$ is in the definition of the circuit P that is obfuscated. In $\mathbf{Exp}^{b,(2)}$ the circuit $P =: P^{(2)}$ is defined as in (10), with $k \leftarrow \mathsf{PF.Smp}(1^\lambda)$. In $\mathbf{Exp}^{b,(3)}$, the key k in circuit $P =: P^{(3)}$ is replaced by a punctured key $k_{h^*} \leftarrow \mathsf{PF.Constr}(k, \{0,1\}^n \setminus \{H(x^*)\})$.

The two games differ thus in whether $\tilde{P}$ is an obfuscation of $P^{(2)}$ or $P^{(3)}$. By public-coin diO, these are indistinguishable, if for a sampler Samp that outputs $P^{(2)}$ and $P^{(3)}$, no extractor, even when given the coins used by Samp, can find a differing input $(\hat{M}, \hat{h}, \hat{\pi}, \hat{\sigma})$.

Suppose there exists an extractor $\mathcal{E}$ outputs such a tuple. By correctness of PF, $P^{(2)}$ and $P^{(3)}$ only differ on inputs $(\hat{M}, \hat{h}, \hat{\pi}, \hat{\sigma})$, where

$$\hat{h} = H(x^*), \tag{12}$$

as that is where the punctured key behaves differently. Moreover, the signature $\hat{\sigma}$ must be valid on $\hat{M}$, as otherwise both circuits output $\perp$. Intuitively, unforgeability of functional signatures should guarantee that

$$\hat{M}(x^*) = 0, \tag{13}$$

as the adversary only obtains a signature from its CONSTR oracle when it submits machines satisfying (13), so a valid $\hat{\sigma}$ on $\hat{M}$ with $\hat{M}(x^*) = 1$ would be a forgery.

To construct $P^{(2)}$ and $P^{(3)}$, Samp must simulate the experiment for $\mathcal{A}$, during which it needs to answer $\mathcal{A}$'s CONSTR queries and thus create signatures. This shows the need for a functional signature scheme: we need to enable Samp to create signatures on M's with $M(x^*) = 0$ (by giving it $sk_{f_{x^*}}$) while still arguing that it is hard to find a signature on $\hat{M}$ with $\hat{M}(x^*) = 1$.

Finally, if we used standard functional signatures then we would need to embed a master verification key (under which the forgery will be) into Samp, but this would require diO *with auxiliary inputs*. We avoid this using FSwOSK, which let Samp create *mvk* (together with sk_{f^*}) itself, and which ensure that for $\mathcal{E}$, even given Samp's coins, it is hard to find a forgery $\hat{\sigma}$. It follows that (13) must hold with overwhelming probability.

Finally the proof $\hat{\pi}$ must be valid for $(H, \hat{M}, \hat{h})$, as otherwise both circuits output $\perp$. By SNARK extractability, we can therefore extract a witness $\hat{x}$ for $(H, \hat{M}, \hat{h}) \in L_{legit}$, that is, (i) $\hat{M}(\hat{x}) = 1$ and (ii) $H(\hat{x}) = \hat{h}$. Now (i) and (13) imply $\hat{x} \neq x^*$ and (ii) and (12) imply $H(\hat{x}) = H(x^*)$. Together, this means $(\hat{x}, x^*)$ is a collision for H.

Overall, we showed that an extractor can only find a differing input for $P^{(2)}$ and $P^{(3)}$ with negligible probability. By security of diO (Definition 4), we thus have that obfuscations of $P^{(2)}$ and $P^{(3)}$ are indistinguishable.

Proposition 3. $\mathbf{Exp}^{b,(2)}$ *and* $\mathbf{Exp}^{b,(3)}$ *are computationally indistinguishable for* $b = 0, 1$*, if* diO *is a public-coin differing-input obfuscator,* FS *a FSwOSK satisfying oblivious unforgeability and* $\mathcal{H}$ *is public-coin collision-resistant.*

For the game hop from games $\mathbf{Exp}^{b,(3)}$ to $\mathbf{Exp}^{b,(4)}$, indistinguishability follows directly from collision resistance of $\mathcal{H}$, as the only difference is that $\mathbf{Exp}^{b,(4)}$ aborts when $\mathcal{A}$ finds a collision.

Proposition 4. $\mathbf{Exp}^{b,(3)}$ *and* $\mathbf{Exp}^{b,(4)}$ *are computationally indistinguishable for* $b = 0, 1$*, if* $\mathcal{H}$ *is CR.*

We have now reached a game, $\mathbf{Exp}^{b,(4)}$, in which the key k is only used to create a punctured key k_{h^*}. The experiment can thus be simulated by an adversary $\mathcal{B}$ against selective security of $\mathcal{PF}$, who first asks for a key for the set $\{0,1\}^n \setminus \{H(x^*)\}$ and then uses $\mathcal{A}$ to distinguish $y^* = \mathsf{PF.Eval}(k, H(x^*))$ from random.

Proposition 5. $\mathbf{Exp}^{0,(4)}$ *and* $\mathbf{Exp}^{1,(4)}$ *are indistinguishable if* $\mathcal{PF}$ *is a selectively secure family of puncturable PRFs.*

Theorem 4 now follows from Propositions 1–5, which are proven in the full version [1].

References

1. Abusalah, H., Fuchsbauer, G.: Constrained PRFs for unbounded inputs with short keys. Cryptology ePrint Archive, Report 2016/279 (2016)
2. Abusalah, H., Fuchsbauer, G., Pietrzak, K.: Constrained PRFs for unbounded inputs. In: Sako, K. (ed.) Topics in Cryptology - CT-RSA 2016. LNCS, vol. 9610, pp. 413–428. Springer, Heidelberg (2016). http://eprint.iacr.org/2014/840
3. Barak, B., Goldreich, O.: Universal arguments and their applications. SIAM J. Comput. **38**(5), 1661–1694 (2008)
4. Bellare, M., Goldwasser, S.: New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 194–211. Springer, Heidelberg (1990)
5. Bitansky, N., Canetti, R., Chiesa, A., Goldwasser, S., Lin, H., Rubinstein, A., Tromer, E.: The hunting of the SNARK. IACR Cryptology ePrint Archive, 2014:580 (2014)
6. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC, pp. 111–120. ACM Press, New York (2013)
7. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 280–300. Springer, Heidelberg (2013)
8. Boneh, D., Waters, B., Zhandry, M.: Low overhead broadcast encryption from multilinear maps. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 206–223. Springer, Heidelberg (2014)
9. Boyle, E., Chung, K.-M., Pass, R.: On Extractability obfuscation. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 52–73. Springer, Heidelberg (2014)
10. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 501–519. Springer, Heidelberg (2014)
11. Fiat, A., Naor, M.: Broadcast encryption. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 480–491. Springer, Heidelberg (1994)
12. Freire, E.S.V., Hofheinz, D., Paterson, K.G., Striecks, C.: Programmable hash functions in the multilinear setting. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 513–530. Springer, Heidelberg (2013)
13. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th FOCS, pp. 40–49. IEEE Computer Society Press (2013)
14. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC, pp. 99–108. ACM Press, June 2011
15. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (1986)
16. Ishai, Y., Pandey, O., Sahai, A.: Public-Coin differing-inputs obfuscation and its applications. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 668–697. Springer, Heidelberg (2015)
17. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: Sadeghi, A.-R., Gligor, V.D., Yung, M. (eds.) ACM CCS 13, pp. 669–684. ACM (2013)

18. Sahai, A., Waters, B., How to use indistinguishability obfuscation: deniable encryption, and more. In: Shmoys, D.B. (ed.) 46th ACM STOC, pp. 475–484. ACM Press, May/June 2014
19. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing. In: SCIS 2000, Okinawa, Japan, January 2000