Functional Signatures and Pseudorandom Functions

Functional Signatures and Pseudorandom Functions

Elette Boyle
MIT

Shafi Goldwasser*†
MIT and Weizmann

Ioana Ivan
MIT

October 29, 2013

Abstract

In this paper, we introduce two new cryptographic primitives: *functional digital signatures* and *functional pseudorandom functions*.

In a functional signature scheme, in addition to a master signing key that can be used to sign any message, there are *signing keys for a function* f, which allow one to sign any message in the range of f. As a special case, this implies the ability to generate keys for predicates P, which allow one to sign any message m, for which $P(m) = 1$.

We show applications of functional signatures to constructing succinct non-interactive arguments and delegation schemes. We give several general constructions for this primitive based on different computational hardness assumptions, and describe the trade-offs between them in terms of the assumptions they require and the size of the signatures.

In a functional pseudorandom function, in addition to a master secret key that can be used to evaluate the pseudorandom function F on any point in the domain, there are additional *secret keys for a function* f, which allow one to evaluate F on any y for which there exists an x such that $f(x) = y$. As a special case, this implies *pseudorandom functions with selective access*, where one can delegate the ability to evaluate the pseudorandom function on inputs y for which a predicate $P(y) = 1$ holds. We define and provide a sample construction of a functional pseudorandom function family for prefix-fixing functions.

This work appeared in part as the Master Thesis of Ioana Ivan filed May 22 at MIT. We note that independently the notion of pseudorandom functions with selective access was studied by Boneh-Waters under the name of *constrained pseudorandom functions* [BW13] and by Kiayias, Papadopoulos, Triandopoulos and Zacharias under the name *delegatable pseudorandom functions* [KPTZ13]. Subsequent to our posting of an earlier manuscript of this work, Bellare and Fuchsbauer [BF13] and Backes, Meiser, and Schröder [BMS13] additionally posted similar results on functional signatures.

*This work was supported in part by Trustworthy Computing: NSF CCF-1018064.

†This material is based on research sponsored by the Air Force Research Laboratory under agreement number FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

1 Introduction

We introduce new cryptographic primitives with a variety of accompanying constructions: *functional digital signatures (FDS)*, *functional pseudorandom functions (F-PRF)*, and *psuedorandom functions with selective access (PRF-SA)*.[1]

Functional Signatures

In digital signature schemes, as defined by Diffie and Hellman [DH76], a signature on a message provides information which enables the receiver to verify that the message has been created by a proclaimed sender. The sender has a secret *signing key*, used in the signing process, and there is a corresponding verification key, which is public and can be used by anyone to verify that a signature is valid. Following Goldwasser, Micali and Rackoff [GMR88], the standard security requirement for signature schemes is unforgeability against chosen-message attack: an adversary that runs in probabilistic polynomial time and is allowed to request signatures for a polynomial number of messages of his choice, cannot produce a signature of any new message with non-negligible probability.

In this work, we extend the classical digital signature notion to what we call *functional signatures*. In a functional signature scheme, in addition to a *master signing key* that can be used to sign any message, there are secondary *signing keys for functions* f (called sk_f), which allow one to sign any message in the range of f. These additional keys are derived from the master signing key. The notion of security we require such a signature scheme to satisfy is that any probabilistic polynomial time (PPT) adversary, who can request signing keys for functions $f_1 \dots f_l$ of his choice, and signatures for messages $m_1, \dots m_q$ of his choice, can only produce a signature of a message m with non-negligible probability, if m is equal to one of the queried messages $m_1, \dots m_q$, or if m is in the range of one of the queried functions $f_1 \dots f_l$.

An immediate application of a functional signature scheme is the ability to delegate the signing process from a master authority to another party. Suppose someone wants to allow their assistant to sign on their behalf only those messages with a certain tag, such as "signed by the assistant". Let P be a predicate that outputs 1 on messages with the proper tag, and 0 on all other messages. In order to delegate the signing of this restricted set of messages, one would give the assistant a signing key for the following function:

$$f(m) := \begin{cases} m \text{ if } P(m) = 1 \\ \perp \text{ otherwise} \end{cases} .$$

P could also be a predicate that checks if the message does not contain a given phrase, or if it is related to a certain subject, or if it satisfies a more complex policy.

Another application of functional signatures is to certify that only allowable computations were performed on data. For example, imagine the setting of a digital camera that produces signed photos (i.e the original photos produced by the camera can be certified). In this case, one may want to allow photo-processing software to perform minor touch-ups of the photos, such as changing the color scale or removing red-eyes, but not allow more significant changes such as merging two photos or cropping a picture. But, how can an original photo which is slightly touched-up be distinguished from one which is the result of a major change? Functional signatures can naturally address this problem by providing the photo processing software with keys which enable it to sign only the allowable modifications of an original photograph. Generalizing, we think of a client and a server (e.g. photo-processing software), where the client provides the server with data (e.g. signed original photos, text documents, medical data) which he wants to be processed in a restricted fashion.A functional signature of the processed data provides proof of allowable processing.

Functional signatures can also be used to construct a delegation scheme. In this setting, there a client who wants to allow a more powerful server to compute a function f on inputs chosen by the client, and

[1] We note that independently (and unknown to the authors) the notion of pseudorandom functions with selective access was studied by Boneh-Waters under the name of *constrained pseudorandom functions* [BW13] and by Kiayias, Papadopoulos, Triandopoulos and Zacharias under the name *delegatable pseudorandom functions* [KPTZ13]. Subsequent to our posting of an earlier manuscript of this work, [BF13] and [BMS13] have additionally posted similar results on functional signatures.

wants to be able to verify that the result returned by the server is correct. The verification process should be more efficient than for the client to compute f himself. The client can give the server a key for the function $f'(x) = (f(x)|x)$. To prove that $y = f(x)$ the prover gives the client a signature of $y|x$, which he could only have obtained if $y|x$ is in the range of f', that is, if $y = f(x)$.

A desirable property of a functional signature scheme is *function privacy*: the signature should reveal neither the function f that the secret key used in the signing process corresponds to, nor the message m that f was applied to. In the example with the signed photos, one might not wish to reveal the original image just that the final photographs were obtained by running one of the allowed functions on some image taken with the camera.

An additional desirable property is *succinctness*: the size of the signature should only depend on the size of the output $f(m)$ and the security parameter (or just the security parameter), rather than the size of the circuit for computing f.

Functional Pseudorandomness

Pseudorandom functions, introduced by Goldreich, Goldwasser, and Micali [GGM86], are a family of indexed functions $F = \{F_s\}$ such that: (1) given the index s, F_s can be efficiently evaluated on all inputs (2) no probabilistic polynomial-time algorithm *without* s can distinguish evaluations $F_s(x_i)$ for inputs x_i's of its choice from random values. Pseudorandom functions are useful for numerous symmetric-key cryptographic applications, including generating passwords, identify-friend-or-foe systems, and symmetric-key encryption secure against chosen ciphertext attacks. In the public-key setting, there is a construction of digital signatures from pseudorandom functions [BG89], via the following paradigm: one may publish a commitment to secret key s and henceforth be able to prove that $y = F_s(x)$ for a pair (x, y) via a non-interactive zero-knowledge (NIZK) proof.

In this work, we extend pseudorandom functions to a primitive which we call *functional pseudorandom functions (F-PRF)*. The idea is that in addition to a master secret key (that can be used to evaluate the pseudorandom function F_s on any point in the domain), there are additional *secret keys* sk_f *per function* f, which allow one to evaluate F_s on any y for which there exists x such that $f(x) = y$ (i.e $y \in Range(f)$). An immediate application of such a construct is to specify succinctly the randomness to be used by parties in a randomized distributed protocol with potentially faulty players, so as to force honest behavior. A centralized authority holds a description of an index s of a pseudorandom function F_s. One may think of this authority as providing a service which dispenses pseudorandomness (alternatively, the secret s can be shared among players in an MPC). The authority provides each party id with a secret key s_{id} which enables party id to (1) evaluate $F_s(y)$ whenever $y =$ "$id\|h$", where h corresponds to say the public history of communication, and (2) use $F_s(y)$ as her next sequence of coins in the protocol. To prove that the appropriate randomness was used, id can utilize NIZK proofs. An interesting open question is how to achieve a *verifiable* F-PRF, where there is additional information vk_s that can be used to verify that a given pair $(x, F_s(x))$ is valid, without assuming the existence of an honestly generated common reference string, as in the NIZK setting. Note that in this example the function $f(x) = y$ is simply the function which appends the string prefix id to x. We note that there are many other ways to force the use of proper randomness in MPC protocols by dishonest parties, starting with the classical paradigm [GM82, GMW86] where parties interact to execute a "coin flip in the well" protocol forcing players to use the results of these coins, but we find the use of F-PRF appealing in its simplicity, lack of interaction and potential efficiency.

The notion of functional pseudorandom functions has many variations. One natural variant that immediately follows is *pseudorandom functions with selective access*: start with a pseudorandom function as defined in [GGM86], and add the ability to generate secondary keys sk_{P_i} (per predicate P_i) which enable computing $F_s(x)$ whenever $P_i(x) = 1$. This is a special case of F-PRF, as we can take the secret key for predicate P_i to be sk_{f_i} where $f_i(x) = x$ if $P_i(x) = 1$ and $\perp$ otherwise. The special case of *punctured PRFs*, in which secondary keys allow computing $F_s(x)$ on all inputs except one, is similarly implied and has recently been shown to have important applications (e.g., [SW13, HSW13]). Another variant is *hierarchical pseudorandom functions*, with an additional property that parties with functional keys sk_f may also generate subordinate

keys sk_g for functions g of the form $g = f \circ f'$ (i.e., first evaluate some function f', then evaluate f). Note that the range of such composition g is necessarily contained within the range of f.

Independent Work. A preliminary version of this work appeared in a Masters Thesis submitted on May 22, 2013. We note that independently (and unknown to the authors) the notion of pseudorandom functions with selective access was studied by Boneh-Waters under the name of *constrained pseudorandom functions* [BW13] and by Kiayias, Papadopoulos, Triandopoulos and Zacharias under the name *delegatable pseudorandom functions* [KPTZ13]. Subsequent to our posting of an earlier manuscript of this work, [BF13] and [BMS13] have additionally posted similar results on functional signatures.

1.1 Our Results on Functional Signatures and Their Applications

We provide a construction of functional signatures achieving function privacy and succinctness, assuming the existence of succinct non-interactive arguments of knowledge (SNARKS) and (standard) non-interactive zero-knowledge arguments of knowledge (NIZKAoKs) for NP languages.

As a building block, we first give a construction of a functional signature scheme that is not succinct or function private, based on a much weaker assumption: the existence of one-way functions.

Theorem 1.1 (Informal)**.** *Based on any one-way function, there exists a functional signature scheme that supports signing keys for any function f computable by a polynomial-sized circuit. This scheme satisfies the unforgeability requirement for functional signatures, but not function privacy or succinctness.*

Overview of the construction: The master signing and verification keys for the functional signature scheme will correspond to a key pair, $(\mathsf{msk}, \mathsf{mvk})$, in an underlying (standard) signature scheme.

To generate a signing key for a function f, we do the following. First, sample a fresh signing and verification key pair $(\mathsf{sk}', \mathsf{vk}')$ in the underlying signature scheme, and sign the concatenation $f|\mathsf{vk}'$ using msk. The signing key for f consists of this signature together with sk'. Given this signing key, a user can sign any message $m^* = f(m)$ by signing m using sk', and outputting this signature, together with the signature of $f|\mathsf{vk}'$ given as part of sk_f.

We then now show how to use a SNARK system, together with this initial construction, to construct a *succinct*, *function-private* functional signature scheme.

A SNARK system for an NP language L with corresponding relation R is an extractable proof system where the size of a proof is sublinear in the size of the witness corresponding to an instance. SNARK schemes have been constructed under various non-falsifiable assumptions. Bitansky et al. [BCCT13] construct zero-knowledge SNARKs where the length of the proof and the verifier's running time are bounded by a polynomial in the security parameter, and the *logarithm* of running time of the corresponding relation $R(x, w)$, assuming the existence of collision resistance hash functions and a knowledge of exponent assumption.[2] (More details are given in Section 2.3.)

Theorem 1.2 (Informal)**.** *Assuming the existence of succinct non-interactive arguments of knowledge (SNARKs), NIZKAoK for NP languages, and a functional signature scheme that is not necessarily function-private or succinct, there exists a succinct, function-private functional signature scheme that supports signing keys for any function f computable by a polynomial-sized circuit.*

Overview of the construction: Our construction makes use of non-succinct, non-function-private functional signature scheme $\mathsf{FS1}$ (which exists based on one-way functions by our construction above), and a zero-knowledge SNARK system for NP.

In the setup algorithm for our functional signature scheme, we sample a key pair $(\mathsf{msk}, \mathsf{mvk})$ for the functional signature scheme $\mathsf{FS1}$, and common reference string crs for the SNARK system. We use msk as

[2] In [BCCT12], Bitansky et al. also show that any SNARK + NIZKAoK directly yield zero-knowledge (ZK)-SNARK with analogous parameters.

the new master singing key and $(\mathsf{mvk}, \mathsf{crs})$ as the new master verification key. The key generation algorithm is the same as in the underlying functional signature scheme $\mathsf{FS1}$. To sign a message m^* using a resulting key sk_f, we generate a zero-knowledge SNARK for the following statement: $\exists\sigma$ such that σ is a valid signature of m^* under mvk in the functional signature scheme $\mathsf{FS1}$. To verify the signature, we run the verification algorithm for the SNARK argument system.

Resorting to non-falsifiable assumptions, albeit strong, seems necessary to obtain succinctness for functional signatures. We show that, given a functional signature scheme with short signatures, we can construct a SNARG system.

Theorem 1.3 (Informal)**.** *If there exists a functional signature scheme supporting keys for all polynomial-sized circuits f, that has short signatures (i.e of size $poly(k) \cdot (|f(m)| + |m|)^{o(1)}$ for security parameter k), then there exists a SNARG scheme with preprocessing for any language $L \in NP$ with proof size $poly(k) \cdot (|w| + |x|)^{o(1)}$, where w is the witness and x is the instance.*

The main idea in the SNARG construction is for the verifier (CRS generator) to give out a single signing key sk_f for a function whose range consists of exactly those strings that are in L. Then, with sk_f, the prover will be able to sign only those messages x that are in the language L, and thus can use this (short) signature as his proof.

Gentry and Wichs showed in [GW11] that SNARG schemes with proof size $poly(k) \cdot (|w| + |x|)^{o(1)}$ cannot be obtained using black-box reductions to falsifiable assumptions. We can thus conclude that in order to obtain a functional signature scheme with signature size $poly(k) \cdot (|f(m)| + |m|)^{o(1)}$ we must either rely on non-falsifiable assumptions (as in our SNARK construction) or make use of non black-box techniques.

Finally, we can construct a scheme which satisfies unforgeability and functional privacy but not succinctness based on the weaker assumption of non-interactive zero-knowledge arguments of knowledge (NIZKAoK) for NP.

Theorem 1.4 (Informal)**.** *Assuming the existence of non-interactive zero-knowledge arguments of knowledge (NIZKAoK) for NP, there exists a functional signature scheme that supports signing keys for any function f computable by a polynomial-sized circuit. This scheme satisfies function privacy, but not succinctness: the size of the signature is dependent on the size of f and m.*

Overview of the construction: The construction is analogous to the SNARK-based construction in the previous construction, with the SNARK system replaced with a NIZKAoK system. Namely, a signature will be a NIZKAoK for the following statement: $\exists\sigma$ such that σ is a valid signature of m^* under mvk, in an underlying non-succinct, non-function-private functional signature scheme, as before (recall such a scheme exists based on OWF). The signature size is now polynomial in the size of σ, which, if $m^* = f(m)$, and sigma was generated using sk_f, is itself polynomial in the security parameter, $|m|$, and $|f|$.

1.1.1 Relation to Delegation:

Functional signatures are highly related to delegation schemes. A delegation scheme allows a client to outsource the evaluation of a function f to a server, while allowing the client to verify the correctness of the computation more efficiently than computing the function himself. We show that given *any* functional signature scheme supporting a class of functions $\mathcal{F}$, we can obtain a delegation scheme in the preprocessing model for functions in $\mathcal{F}$, with related parameters.

Theorem 1.5 (Informal)**.** *If there exists a functional signature scheme for function class $\mathcal{F}$, with signature size $s(k)$, and verification time $t(k)$, then there exists a one-round delegation scheme for functions in $\mathcal{F}$, with server message size $s(k)$ and client verification time $t(k)$.*

Overview of the construction: The client can give the server a key $\mathsf{sk}_{f'}$ for the function $f'(x) = (f(x)|x)$. To prove that $y = f(x)$, the prover gives the client a signature of $y|x$, which he could only have obtained if $y|x$ is in the range of f'; that is, if $y = f(x)$. The length of a proof is equal to the length of a signature

in the functional signature scheme, $s(k)$, and the verification time for the delegation scheme is equal to the verification time of the functional signature scheme.

1.2 Summary of our Results on Functional Pseudorandom Functions and Selective Pseudorandom Functions

We present formal definitions and constructions of functional pseudorandom functions (F-PRF) and pseudorandom functions with selective access (PRF-SA). In particular, we present a construction based on the existence of one-way functions of a functional pseudorandom function family supporting the class of *prefix -fixing functions*. Our construction is based on the Goldreich-Goldwasser-Micali (GGM) tree-based PRF construction [GGM86].

Theorem 1.6 (Informal)**.** *Assuming the existence of OWF, there exists a functional PRF that supports keys for the following class of functions related to prefix matching:* $\mathcal{F}_{\mathsf{pre}} = \{f_z | z \in \{0,1\}^m, m \leq n\}$*, where* $f_z(x) = x$ *if* z *is a prefix of* x*, and* $\perp$ *otherwise. The pseudorandomness property holds against a selective adversary, who declares the functions he will query before seeing the public parameters.*

We remark that one can directly obtain a *fully* secure F-PRF for $\mathcal{F}_{\mathsf{pre}}$, in which security holds against an adversary who adaptively requests key queries, from our selectively secure construction, with a loss of 2^{-n} in security for each functional secret key sk_{f_z} queried by the adversary. This is achieved simply by guessing the adversary's query $f_z \in \mathcal{F}_{\mathsf{pre}}$. For appropriate choices of the input length n, security of the underlying OWF, and number of key queries, this still provides the required security.

Overview of the construction. We show that the original Goldreich-Goldwasser-Micali (GGM) tree-based construction [GGM86] provides the desired functionality, where the functional key sk_f corresponding to a prefix-fixing function $f_z(x) = z_1 z_2 \cdots z_i x_{i+1} \cdots x_n$ will be given by the partial evaluation of the PRF down the tree, at the node corresponding to prefix $z_1 z_2 \cdots z_i$.

This partial evaluation clearly enables a user to compute all possible continuations in the evaluation tree, corresponding to the output of the PRF on any input possessing prefix z. Intuitively, security holds since the other partial evaluations at this level i in the tree still appear random given the evaluation sk_f (indeed, this corresponds to a truncated i-bit input GGM construction).

Punctured pseudorandom functions. Punctured pseudorandom functions [SW13] are a special case of functional PRFs where one can generate keys for the function family $\mathcal{F} = \{f_x(y) = y$ if $y \neq x$, and $\perp$ otherwise$\}$. Namely, a key for function f_x allows one to compute the pseudorandom function on any input except for x. Punctured PRFs have recently proven useful as one of the main techniques used in proving the security of various cryptographic primitives based on the existence of indistinguishability obfuscation. Some examples include a construction of public-key encryption from symmetric-key encryption and the construction of deniable encryption given by Sahai and Waters in [SW13], as well as an instantiation of random oracles with a concrete hash function for full-domain hash applications by Hohenberger et al. in [HSW13].

We note that the existence of a functional PRF for the prefix-fixing function family gives a construction of punctured PRFs. A key that allows one to compute the PRF on all inputs except $x = x_1 \ldots x_n$ consists of n functional keys for the prefix-fixing function family for prefixes: $\bar{x_1}$, $x_1\bar{x_2}$, $x_1x_2\bar{x_3} \ldots x_1x_2 \ldots x_{n-1}\bar{x_n}$. We remark that while n prefix-matching keys are revealed, there are only 2^n such *sets* of keys (corresponding to the 2^n choices for the punctured input x), and thus we lose only 2^{-n} security when complexity leveraging from selective to full security. For appropriate choice of underlying OWF security, this yields fully secure punctured PRFs for any desired poly-sized inputs, based on OWFs.

Corollary 1.7. *Assuming the existence of OWF, there exists a (fully) secure punctured PRF for any desired poly-size input length.*

Other notions. Our construction has the additional beneficial property of *hierarchical key generation*: i.e., a party with a functional key sk_{f_z} for a prefix z may generate valid "subordinate" functional keys $\mathsf{sk}_{f_{z'}}$ for any prefix $z' = z|*$. That is, we prove the following additional statement.

Corollary 1.8 (Informal)**.** *Assuming the existence of OWF, there exists a* hierarchical *functional PRF for the class of functions* F_{pre}*.*

Recall that we can also view the prefix-matching function as a predicate allowing only signatures of message that begin with a prefix z. As an immediate corollary of the above, we achieve (hierarchical) functional PRFs with *selective access* for the corresponding class of prefix-matching predicates:

Corollary 1.9 (Informal)**.** *Assuming the existence of OWF, there exists a (hierarchical) functional PRF with selective access for the class of prefix-matching predicates* $\mathcal{P}_{\mathsf{pre}} = \{P_z | z \in \{0,1\}^m, m \leq n\}$*, where* $P_z(x) = 1$ *if* z *is a prefix of* x*, and* 0 *otherwise. The pseudorandomness property holds against a selective adversary (or against an adaptive adversary, with a security loss of* 2^{-n} *per key query).*

1.3 Open Problems

The size of the signatures in our SNARK-based functional signature scheme is dependent only of the security parameter, but it is based on non-falsifiable assumptions. In Section 4, we show that, for a functional signature scheme that supports signing keys for a function f, a signature of $y = f(x)$ cannot be sublinear in the size of y or x, unless the construction is either proven secure under a non-falsifiable assumption or makes use of non blac-kbox techniques. No lower bound exists that relates the size of the signature to the description of f. Constructing functional signatures with short (sublinear in the size of the functions supported) signatures and verification time under falsifiable assumptions remains an open problem.

An interesting problem left open by this work is to construct a functional PRF that is also *verifiable*. A verifiable PRF, introduced by Micali, Rabin and Vadhan in [MRV99] has the property that, in addition to the secret seed of the PRF, there is a corresponding public key and a way to generate a proof π_x given the secret seed, such that given the public key, x, y and π_x one can check that y is indeed the output of the PRF on x. The public parameters and the proof should not allow an adversary to distinguish the outputs of the PRF from random on any point for which the adversary has not received a proof. A construction of standard verifiable PRFs was given by Lysyanskaya based on the many-DH assumption in bilinear groups in [Lys02].

One may extend the notion of verifiable PRFs to the setting of functional PRFs by enabling a user with functional key sk_f to also generate verifiable proofs π_x of correctness for evaluations of the PRF on inputs x for which his key allows. We note that such a verifiable functional pseudorandom function family supporting keys for a function class $\mathcal{F}$, implies a functional signature scheme that supports signing keys for the same function class, so the lower bound mentioned for functional signatures applies also to the proofs output in the verifiable functional PRF context.

1.4 Other Related Work

Functional Encryption. This work is inspired by recent results on the problem of functional encryption, which was introduced by Sahai and Waters in [SW05], and formalized by Boneh et al. in [BSW11]. In the past few years there has been significant progress on constructing functional encryption schemes for general classes of functions (e.g., [GVW12, GKP$^+$12, GKP$^+$13]). In this setting, a party with access to a master secret key can generate secret keys for any function f, which allows a third party who has this secret key and an encryption of a message m to learn $f(m)$, but nothing else about m. In [GKP$^+$12], Goldwasser et al. construct a functional encryption scheme that can support general functions, where the ciphertext size grows with the maximum depth of the functions for which keys are given. They improve this result in a follow-up work [GKP$^+$13], which constructs a functional encryption scheme that supports decryption keys for any Turing machine. Both constructions are secure according to a simulation-based definition, as long as a single key is given out. In [AGVW13], Agrawal et al. show that constructing functional encryption schemes

achieving this notion of security in the presence of an unbounded number of secret keys is impossible for general functions. In contrast, no such impossibility results are known in the setting of functional signatures.

Connections to Obfuscation. The goal of program obfuscation is to construct a compiler O that takes as input a program P and outputs a program $O(P)$ that preserves the functionality of P, but hides all other information about the original program. In [BGI+01] Barak et al. formalize this, requiring that, for every adversary having access to an obfuscation of P that outputs a single bit, there exists a simulator that only has blackbox access to P and whose output is statistically close to the adversary's output:

$$\Pr[A(O(P)) = 1] - \Pr[S^P(1^{|P|}) = 1] = \mathsf{negl}(|P|)$$

Barak et al. [BGI+01] construct a class of programs and an adversary for which no simulator can exist, therefore showing that this definition is not achievable for general functions. Furthermore, in [GK05], Goldwasser and Kalai give evidence that several natural cryptographic algorithms, including the signing algorithm of any unforgeable signature scheme, are not obfuscatable with respect to this strong definition.

Consider the function $\mathsf{Sign} \circ f$, where Sign is the signing algorithm of an unforgeable signature scheme, f is an arbitrary function and $\circ$ denotes function composition. Based on the results in [GK05] we would expect this function not to be obfuscatable according to the blackbox simulation definition. A meaningful relaxation of the definition is that, while having access to an obfuscation of this function might not hide all information about the signing algorithm, it does not completely reveal the secret key, and does not allow one to sign messages that are not in the range of f. In our function signature scheme, the signing key corresponding to a function f achieves exactly this definition of security, and we can think of it as an obfuscation of $\mathsf{Sign} \circ f$ according to this relaxed definition. Indeed it has recently come to our attention that Barak in an unpublished manuscript has considered *delegatable signatures*, a highly related concept.

Homomorphic Signatures. Another related problem is that of homomorphic signatures. In a homomorphic signature scheme, a user signs several messages with his secret key. A third party can then perform arbitrary computations over the signed data, and obtain a new signature that authenticates the resulting message with respect to this computation. In [GW12], Gennaro and Wichs construct homomorphic message authenticators, which satisfy a weaker unforgeability notion than homomorphic signatures, in that the verification is done with respect to a secret key unknown to the adversary. They impose an additional restriction on the adversary, who is not allowed to make verification queries. For homomorphic signature schemes with public verification, the most general construction of Boneh and Freeman [BF11] only allows the evaluation of multivariate polynomials on signed data. Constructing homomorphic signature schemes for general functions remains an open problem.

Signatures of correct computation. Papamanthou, Shi and Tamassia considered a notion of functional signatures under the name "signatures of correct computation" in [PST13]. They give constructions for schemes that support operations over multivariate polynomials, such as polynomial evaluation and differentiation. Their constructions are secure in the random oracle model and allow efficient updates to the signing keys: the keys can be updated in time proportional to the number of updated coefficients. In contrast, our constructions that support signing keys for general functions, assuming the existence of succinct non-interactive arguments of knowledge.

Independent work. Finally, as mentioned earlier, related notions to functional PRFs appear in the concurrent and independent works [BW13, KPTZ13]. Based on the Multilinear Decisional Diffie-Hellman assumption (a recently coined assumption related to existence of secure multilinear maps), [BW13] show that PRFs with Selective Access can be constructed for all predicates describable as polynomial-sized circuits. We remark that this is not equivalent to functional PRFs for polynomial-sized circuits, which additionally captures NP relations (i.e., the predicate $y \in \mathit{Range}(f)$ may not be efficiently testable directly).

Subsequent to our posting of an earlier manuscript of this work, [BF13] and [BMS13] have additionally posted similar results on functional signatures.

1.5 Overview of the paper

In Section 2, we describe several primitives which will be used in our constructions. In Section 3, we give a formal definition of functional signature schemes, and present three constructions satisfying the definition. In Section 4, we show how to construct delegation schemes and succinct non-interactive arguments (SNARGs) from functional signatures schemes. In Section 5, we give a formal definition of functional pseudorandom functions and pseudorandom functions with selective access, and present a sample construction for the prefix-fixing function family.

2 Preliminaries

In this section we define several cryptographic primitives that are used in our constructions.

2.1 Signature Schemes

Definition 2.1. A signature scheme for a message space $\mathcal{M}$ is a tuple $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$:

- $\mathsf{Gen}(1^k) \to (\mathsf{sk}, \mathsf{vk})$: the key generation algorithm is a probabilistic, polynomial-time algorithm which takes as input a security parameter 1^k, and outputs a signing and verification key pair $(\mathsf{sk}, \mathsf{vk})$.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: the signing algorithm is a probabilistic polynomial time algorithm which is given the signing key sk and a message $m \in \mathcal{M}$ and outputs a string σ which we call the signature of m.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to \{0, 1\}$: the verification algorithm is a polynomial time algorithm which, given the verification key vk, a message m, and signature σ, returns 1 or 0 indicating whether the signature is valid.

A signature scheme should satisfy the following properties:
Correctness

$$\forall (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k), \forall m \in \mathcal{M}, \forall \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m),$$
$$\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to 1$$

Unforgeability under chosen message attack
A signature scheme is unforgeable under chosen message attack if the winning probability of any probabilistic polynomial time adversary in the following game is negligible in the security parameter:

- The challenger samples a signing, verification key pair $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k)$ and gives vk to the adversary.
- The adversary requests signatures from the challenger for a polynomial number of messages. In round i, the adversary chooses m_i based on $m_1, \sigma_1, \ldots m_{i-1}, \sigma_{i-1}$, and receives $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$.
- The adversary outputs a signature σ^* and a message m^*, and wins if $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) \to 1$ and the adversary has not previously received a signature of m^* from the challenger.

Lemma 2.2 ([Rom90])**.** *Under the assumption that one-way functions exist, there exists a signature scheme which is secure against existential forgery under adaptive chosen message attacks by polynomial-time algorithms.*

2.2 Non-Interactive Zero Knowledge

Definition 2.3. [FLS90, BFM88, BSMP91]: $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}))$ is an *efficient adaptive NIZK argument system* for a language $L \in \mathsf{NP}$ with witness relation $\mathcal{R}$ if $\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}$ are all PPT algorithms, and there exists a negligible function μ such that for all k the following three requirements hold.

- **Completeness**: For all x, w such that $\mathcal{R}(x, w) = 1$, and for all strings $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$,

$$\mathsf{Verify}(\mathsf{crs}, x, \mathsf{Prove}(x, w, \mathsf{crs})) \to 1.$$

- **Adaptive Soundness**: For all PPT adversaries $\mathcal{A}$, if $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$ is sampled uniformly at random, then the probability that $\mathcal{A}(\mathsf{crs})$ will output a pair (x, π) such that $x \notin L$ and yet $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to 1$, is at most $\mu(k)$.
- **Adaptive Zero-Knowledge**: For all PPT adversaries $\mathcal{A}$,

$$\left| \Pr[\mathsf{Exp}_{\mathcal{A}}(k) \to 1] - \Pr[\mathsf{Exp}^{S}_{\mathcal{A}}(k) \to 1] \right| \leq \mu(k),$$

  where the experiment $\mathsf{Exp}_{\mathcal{A}}(k)$ is defined by:

$$\begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^k) \\ \text{Return } \mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) \end{array}$$

  and the experiment $\mathsf{Exp}^{S}_{\mathcal{A}}(k)$ is defined by:

$$\begin{array}{l} (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k) \\ \text{Return } \mathcal{A}^{S'(\mathsf{crs},\mathsf{trap},\cdot,\cdot)}(\mathsf{crs}), \end{array}$$

  where $S'(\mathsf{crs}, \mathsf{trap}, x, w) = \mathcal{S}^{\mathsf{Proof}}(\mathsf{crs}, \mathsf{trap}, x)$.

We next define the notion of a NIZK argument of knowledge.

Definition 2.4. Let $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}))$ be an efficient adaptive NIZK argument system for an NP language $L \in \mathsf{NP}$ with a corresponding NP relation $\mathcal{R}$. We say that Π is a *argument-of-knowledge* if there exists a PPT algorithm $\mathsf{E} = (\mathsf{E}_1, \mathsf{E}_2)$ such that for *every* adversary $\mathcal{A}$,

$$|\Pr[\mathcal{A}(\mathsf{crs}) \to 1 | \mathsf{crs} \leftarrow \mathsf{Gen}(1^k)] - \Pr[\mathcal{A}(\mathsf{crs}) \to 1 | (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^k)]| = \mathsf{negl}(k)$$

For every PPT adversary $\mathcal{A}$,

$$\begin{array}{c} \Pr[\mathcal{A}(\mathsf{crs}) \to (x, \pi) \text{ and } \mathsf{E}(\mathsf{crs}, \mathsf{trap}, x, \pi) \to w^* \text{ s.t. } \mathsf{Verify}(\mathsf{crs}, x, \pi) \to 1 \text{ and } (x, w^*) \notin \mathcal{R}] \\ = \mathsf{negl}(k), \end{array}$$

where the probabilities are taken over $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^k)$, and over the random coin tosses of the extractor algorithm E_2.

We note that we require the distributions over the honestly generated crs, and the crs generated by the extractor E_1 to be statistically close, whereas they are often required to be just computationally indistinguishable. However, if one is satisfied with computational zero knowledge (as is the case for us), this is actually without loss of generality. Namely, given any NIZKAoK $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}), \mathsf{E} = (\mathsf{E}_1, \mathsf{E}_2))$ for which the CRS output by $\mathsf{Gen}(1^k)$ and $\mathsf{E}_1(1^k)$ are only computationally indistinguishable, we claim that the system Π' formed by using E_1 also as the *honest* CRS generation algorithm (i.e., replacing Gen) is also a NIZKAoK, and satisfies our statistical indistinguishability requirement.

Claim 2.5. *Suppose Π as above is a NIZKAoK for which $\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{Gen}(1^k)\} \stackrel{c}{\cong} \{\mathsf{crs} : (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^k)\}$ are only computationally indistinguishable. Then $\Pi' := (\mathsf{E}_1, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}), \mathsf{E} = (\mathsf{E}_1, \mathsf{E}_2))$ is a NIZKAoK as in Definition 2.4.*

Proof. Clearly extraction and adaptive soundness are maintained. Completeness must still hold for any statement/witness pairs (x, w) produced by an efficient adversary, due to the computational indistinguishability of CRSs generated by Gen and E_1; otherwise there exists an efficient distinguishing algorithm who generates honest proofs and tests whether they verify. Finally, adaptive zero knowledge must hold for the *same* simulator algorithms $\mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}})$. Indeed, for PPT adversary $\mathcal{A}$, consider a third experiment $\mathsf{Exp}^{\mathsf{E}}_{\mathcal{A}}(k)$ defined by generating $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^k)$, and returning $\mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs})$. That is, $\mathsf{Exp}^{\mathsf{E}}_{\mathcal{A}}(k)$ is identical to the real-world experiment $\mathsf{Exp}_{\mathcal{A}}(k)$ except that the CRS is generated according to E_1 instead of Gen (and, in particular, corresponds to the real-world experiment for the modified scheme Π'). By the computational indistinguishability of CRSs generated by Gen and E_1, it holds that $\mathsf{Exp}_{\mathcal{A}}(k)$ and $\mathsf{Exp}^{\mathsf{E}}_{\mathcal{A}}(k)$ are computationally indistinguishable. But by the adaptive zero knowledge property of the original scheme Π, we have that $\mathsf{Exp}_{\mathcal{A}}(k)$ is computationally indistinguishable from the simulated experiment $\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}(k)$. Thus, it must be that $\mathsf{Exp}^{\mathsf{E}}_{\mathcal{A}}(k)$ is computationally indistinguishable from $\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}(k)$: that is, Π' satisfies adaptive zero knowledge. □

Remark. There is a standard way to convert any NIZK argument system Π to a NIZK argument-of-knowledge system Π'. The idea is to append to the crs a public key pk corresponding to any semantic secure encryption scheme. Thus, the common reference string corresponding to Π' is of the form $\mathsf{crs}' = (\mathsf{crs}, \mathsf{pk})$. In order to prove that $x \in L$ using a witness w, choose randomness $r \leftarrow \{0,1\}^{\mathsf{poly}(k)}$, compute $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(w, r)$ and compute a NIZK proof π, using the underlying NIZK argument system Π, that $(\mathsf{pk}, x, c) \in L'$, where

$$L' \triangleq \{(\mathsf{pk}, x, c) : \exists (w, r) \text{ s.t. } (x, w) \in \mathcal{R} \text{ and } c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(w, r)\}.$$

Let $\pi' = (\pi, c)$ be the proof.

The common reference string simulator E_1 will generate a simulated crs' by generating $(\mathsf{crs}, \mathsf{trap})$ using the underlying simulator $\mathcal{S}^{\mathsf{crs}}$, and by generating a public key pk along with a corresponding secret key sk. Thus, $\mathsf{trap}' = (\mathsf{trap}, \mathsf{sk})$. The extractor algorithm E_2, will extract a witness for x from a proof $\pi' = (\pi, c)$ by using sk to decrypt the ciphertext c.

We note that the distribution over the honestly generated crs, and the crs generated by E_1 are statistically close, as required in our definition above.

Lemma 2.6 ([FLS90])**.** *Assuming the existence of enhanced trapdoor permutations, there exists an efficient adaptive NIZK argument of knowledge for all languages in NP.*

2.3 Succinct Non-Interactive Arguments (SNARGs)

Definition 2.7. $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ is a *succinct non-interactive argument* for a language $L \in \mathsf{NP}$ with witness relation $\mathcal{R}$ if it satisfies the following properties:

- **Completeness**: For all x, w such that $\mathcal{R}(x, w) = 1$, and for all strings $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$,

$$\mathsf{Verify}(\mathsf{crs}, x, \mathsf{Prove}(x, w, \mathsf{crs})) = 1.$$

- **Adaptive Soundness**: There exists a negligible function $\mu(k)$, such that, for all PPT adversaries $\mathcal{A}$, if $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$ is sampled uniformly at random, then the probability that $\mathcal{A}(\mathsf{crs})$ will output a pair (x, π) such that $x \notin L$ and yet $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$, is at most $\mu(k)$.
- **Succinctness**: There exists an universal polynomial $p(\cdot)$ that does not depend on the relation $\mathcal{R}$, such that

$$\forall x, w \text{ s.t } R(x, w) = 1, \mathsf{crs} \leftarrow \mathsf{Gen}(1^k), \pi \leftarrow \mathsf{Prove}(x, w, \mathsf{crs}),$$
$$|\pi| \leq p(k + \log R)$$

  where R denotes the runtime of the relation associated with language L. We note that the definition of succinctness considered in the lower bound of [GW11] is weaker, in that they require the proof size to only be bounded by $r(k) \cdot (|x| + |w|)^{o(1)}$, for some polynomial $r(\cdot)$.

Definition 2.8. A SNARG $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ is a *succinct non-interactive argument of knowledge(SNARK)* for a language $L \in \mathsf{NP}$ with witness relation $\mathcal{R}$ if there exists a negligible function $\mu(\cdot)$ such that, for all PPT provers P^*, there exists a PPT algorithm $\mathsf{E}_{P^*} = (\mathsf{E}^1{}_{P^*}, \mathsf{E}^2{}_{P^*})$ such that for *every* adversary $\mathcal{A}$,

$$|\Pr[\mathcal{A}(\mathsf{crs}) \to 1 | \mathsf{crs} \leftarrow \mathsf{Gen}(1^k)] - \Pr[\mathcal{A}(\mathsf{crs}) \to 1 | (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}^1{}_{P^*}(1^k)]| = \mu(k),$$

and,

$$\Pr[P^*(\mathsf{crs}) \to (x, \pi) \text{ and } \mathsf{E}^2{}_{P^*}(\mathsf{crs}, \mathsf{trap}, x, \pi) \to w^* \text{ s.t. } \mathsf{Verify}(\mathsf{crs}, x, \pi) \to 1 \text{ and } (x, w^*) \notin \mathcal{R}]$$
$$= \mu(k).$$

where the probabilities are taken over $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}^1_{P^*}(1^k)$, and over the random coin tosses of the extractor algorithm $\mathsf{E}^2_{P^*}$.

Remark As in the NIZK definition, we require the distributions over the honestly generated crs, and the crs generated by the extractor $\mathsf{E}^1_{\mathsf{P}^*}$ to be statistically close. We note that the SNARK construction in [BCCT13] satisfies a stronger definition, where the extraction process has to work for a honestly generated crs, without having access to a trapdoor.

Definition 2.9. A SNARK $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{E})$ is a *zero-knowledge SNARK* for a language $L \in \mathsf{NP}$ with witness relation $\mathcal{R}$ if there exist PPT algorithms$S = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}})$ satisfing the following property:
Adaptive Zero-Knowledge: For all PPT adversaries $\mathcal{A}$,

$$\left| \Pr[\mathsf{Exp}_{\mathcal{A}}(k) \to 1] - \Pr[\mathsf{Exp}^S_{\mathcal{A}}(k) \to 1] \right| \leq \mu(k),$$

where the experiment $\mathsf{Exp}_{\mathcal{A}}(k)$ is defined by:

$$\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$$
$$\text{Return } \mathcal{A}^{\mathsf{Prove}(\mathsf{crs}, \cdot, \cdot)}(\mathsf{crs})$$

and the experiment $\mathsf{Exp}^S_{\mathcal{A}}(k)$ is defined by:

$$(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)$$
$$\text{Return } \mathcal{A}^{S'(\mathsf{crs}, \mathsf{trap}, \cdot, \cdot)}(\mathsf{crs}),$$

where $S'(\mathsf{crs}, \mathsf{trap}, x, w) = \mathcal{S}^{\mathsf{Proof}}(\mathsf{crs}, \mathsf{trap}, x)$.

There are several constructions of SNARKs known, all based on non-falsifiable assumptions. A falsifiable assumption is an assumption that can be modeled as a game between an efficient challenger and an adversary. Most standard cryptographic assumptions are falsifiable. This includes both general assumptions like the existence of OWFs, trapdoor predicates, and specific assumptions (discrete logarithm, RSA, LWE, hardness of factoring).

Lemma 2.10 ([BCCT13])**.** *A SNARK system for any language $L \in NP$ can be constructed assuming the existence of collision-resistant hash function and knowledge of exponent assumptions.*

Lemma 2.11 ([BCCT12])**.** *If there exist SNARKs and NIZKAoK for NP, then there exist zero-knowledge SNARKs for all languages in NP.*

In [GW11] Gentry and Wichs show that no construction of SNARGs, with proof size bounded by $r(k) \cdot (|x| + |w|)^{o(1)}$, for some polynomial $r(\cdot)$, can be proved secure under a black-box reduction to a falsifiable assumption. A black-box reduction is one that only uses oracle access to an attacker, and does not use that adversary's code in any other way. The definition of succinctness in [GW11] is a relaxation of the one in definition Definition 2.8, which makes their lower bound result stronger.

2.4 Delegation Schemes

A delegation scheme allows a client to outsource the evaluation of a function F to a server, while allowing the client to verify the correctness of the computation. The verification process should be more efficient than computing the function. We formalize these requirements below, following the definition introduced by Gennaro et al. in [GGP10].

Definition 2.12 ([GGP10])**.** A *delegation scheme* for a function F consists of a tuple of algorithms (KeyGen, Encode, $\mathsf{Compute}$, Verify)

- $\mathsf{KeyGen}(1^k, F) \rightarrow (\mathsf{enc}, \mathsf{evk}, \mathsf{vk})$: The key generation algorithm takes as input a security parameter k and a function F, and outputs a key enc that is used to encode the input, an evaluation key evk that is used for the evaluation of the function F, and a verification key vk that is used to verify that the output was computed correctly.
- $\mathsf{Encode}(\mathsf{enc}, x) \rightarrow \sigma_x$: The encoding algorithm uses the encoding key enc to encode the function input x as a public value σ_x, which is given to the server to compute with.
- $\mathsf{Compute}(\mathsf{evk}, \sigma_x) \rightarrow (y, \pi_y)$: Using the public evaluation key, evk and the encoded input σ_x, the server computes the function output y = F(x), and a proof π_y that y is the correct output.
- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi_y) \rightarrow \{0, 1\}$: The verification algorithm checks the proof π_y and outputs 1(indicating that the proof is correct), or 0 otherwise.

We require a delegation scheme to satisfy the following requirements:

Correctness
For all vk, x, y, π_y such that $(\mathsf{enc}, \mathsf{evk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^k, F)$, $\sigma_x \leftarrow \mathsf{Encode}(\mathsf{enc}, x)$, $(y, \pi_y) \leftarrow \mathsf{Compute}(\mathsf{evk}, \sigma_x)$,

$$\mathsf{Verify}(\mathsf{vk}, x, y, \pi_y) \rightarrow 1$$

Authentication
For all PPT adversaries, the probability that the adversary is successful in the following game is negligible:

- The challenger runs $\mathsf{KeyGen}(1^k, F) \rightarrow (\mathsf{enc}, \mathsf{evk}, \mathsf{vk})$, and gives $(\mathsf{evk}, \mathsf{vk})$ to the adversary.
- The adversary gets access to an encoding oracle, $O_{enc}(\cdot) = \mathsf{Encode}(\mathsf{enc}, \cdot)$.
- The adversary is successful if it can produce a tuple (x, y, π_y) such that $y \neq F(x)$ and $\mathsf{Verify}(\mathsf{vk}, x, y, \pi_y) \rightarrow 1$.

Efficient verification
Let $T(n)$ be the running time of the verification algorithm on inputs of size n. Let $T_F(n)$ be the running time of F on inputs of size n. We require the worst-case running time of the verification algorithm to be sub linear in the worst case running time of F,

$$T(n) \in o(T_F(n))$$

2.5 Pseudorandom Generators and Functions

Definition 2.13. A *pseudorandom generator* (PRG) is a length expanding function $\mathsf{prg} : \{0,1\}^k \rightarrow \{0,1\}^n$ (for $n > k$) such that $\mathsf{prg}(U_k)$ and U_n are computationally indistinguishable, where U_k is a uniformly distributed k-bit string and U_n is a uniformly distributed n-bit string.

Definition 2.14. [GGM86] A family of functions $\mathcal{F} = \{F_s\}_{s \in S}$, indexed by a set S, and where $F_s : D \rightarrow R$ for all s, is a *pseudorandom function (PRF) family* if for a randomly chosen s, and all PPT $\mathcal{A}$, the distinguishing advantage $\Pr_{s \leftarrow S}[\mathcal{A}^{f_s(\cdot)} = 1] - \Pr_{f \leftarrow (D \rightarrow R)}[\mathcal{A}^{\rho(\cdot)} = 1]$ is negligible, where $(D \rightarrow R)$ denotes the set of *all* functions from D to R.

3 Functional Signatures: Definition and Constructions

3.1 Formal Definition

We now give a formal definition of a functional signature scheme, and explain in more detail the unforgeability and function privacy properties a functional signature scheme should satisfy.

Definition 3.1. A *functional signature scheme* for a message space $\mathcal{M}$, and function family $\mathcal{F} = \{f : \mathcal{D}_f \to \mathcal{M}\}$ consists of algorithms ($\mathsf{FS.Setup}$, $\mathsf{FS.KeyGen}$, $\mathsf{FS.Sign}$, $\mathsf{FS.Verify}$):

- $\mathsf{FS.Setup}(1^k) \to (\mathsf{msk}, \mathsf{mvk})$: the setup algorithm takes as input the security parameter and outputs the master signing key and master verification key.
- $\mathsf{FS.KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$: the key generation algorithm takes as input the master signing key and a function $f \in \mathcal{F}$ (represented as a circuit), and outputs a signing key for f.
- $\mathsf{FS.Sign}(f, \mathsf{sk}_f, m) \to (f(m), \sigma)$: the signing algorithm takes as input the signing key for a function $f \in \mathcal{F}$ and an input $m \in \mathcal{D}_f$, and outputs $f(m)$ and a signature of $f(m)$.
- $\mathsf{FS.Verify}(\mathsf{mvk}, m^*, \sigma) \to \{0, 1\}$: the verification algorithm takes as input the master verification key mvk, a message m and a signature σ, and outputs 1 if the signature is valid.

We require the following conditions to hold:

Corectness:
$\forall f \in \mathcal{F}, \forall m \in \mathcal{D}_f, (\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{FS.Setup}(1^k), \mathsf{sk}_f \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f), (m^*, \sigma) \leftarrow \mathsf{FS.Sign}(f, \mathsf{sk}_f, m)$,

$$\mathsf{FS.Verify}(\mathsf{mvk}, m^*, \sigma) = 1.$$

Unforgeability:
The scheme is unforgeable if the advantage of any PPT algorithm A in the following game is negligible:

- The challenger generates $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{FS.Setup}(1^k)$, and gives mvk to A
- The adversary is allowed to query a key generation oracle $\mathsf{O_{key}}$, and a signing oracle $\mathsf{O_{sign}}$, that share a dictionary indexed by tuples $(f, i) \in \mathcal{F} \times \mathbb{N}$, whose entries are signing keys: $\mathsf{sk}_f^i \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f)$. This dictionary keeps track of the keys that have been previously generated during the unforgeability game. The oracles are defined as follows :
  - $\mathsf{O_{key}}(f, i)$:
    - if there exists an entry for the key (f, i) in the dictionary, then output the corresponding value, sk_f^i.
    - otherwise, sample a fresh key $\mathsf{sk}_f^i \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f)$, add an entry $(f, i) \to \mathsf{sk}_f^i$ to the dictionary, and output sk_f^i
  - $\mathsf{O_{sign}}(f, i, m)$:
    - if there exists an entry for the key (f, i) in the dictionary, then generate a signature on $f(m)$ using this key: $\sigma \leftarrow \mathsf{FS.Sign}(f, \mathsf{sk}_f^i, m)$.
    - otherwise, sample a fresh key $\mathsf{sk}_f^i \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f)$, add an entry $(f, i) \to \mathsf{sk}_f^i$ to the dictionary, and generate a signature on $f(m)$ using this key: $\sigma \leftarrow \mathsf{FS.Sign}(f, \mathsf{sk}_f^i, m)$.
- The adversary wins if it can produce (m^*, σ) such that
  - $\mathsf{FS.Verify}(\mathsf{mvk}, m^*, \sigma) = 1$.
  - there does not exist m such that $m^* = f(m)$ for any f which was sent as a query to the $\mathsf{O_{key}}$ oracle.
  - there does not exist a (f, m) pair such that (f, m) was a query to the $\mathsf{O_{sign}}$ oracle and $m^* = f(m)$.

Function privacy:
Intuitively, we require the distribution of signatures on a message m' generated via different keys sk_f to be computationally indistinguishable, *even given the secret keys and master signing key.* Namely, the advantage of any PPT adversary in the following game is negligible:

- The challenger honestly generates a key pair $(\mathsf{mvk}, \mathsf{msk}) \leftarrow \mathsf{FS.Setup}(1^k)$ and gives both values to the adversary. (Note wlog this includes the randomness used in generation).
- The adversary chooses a function f_0 and receives an (honestly generated) secret key $\mathsf{sk}_{f_0} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f_0)$.
- The adversary chooses a second function f_1 for which $|f_0| = |f_1|$ (where padding can be used if there is a known upper bound) and receives an (honestly generated) secret key $\mathsf{sk}_{f_1} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f_1)$.
- The adversary chooses a pair of values m_0, m_1 for which $|m_0| = |m_1|$ and $f_0(m_0) = f_1(m_1)$.
- The challenger selects a random bit $b \leftarrow \{0,1\}$ and generates a signature on the image message $m' = f_0(m_0) = f_1(m_1)$ *using secret key* sk_{f_b}, and gives the resulting signature $\sigma \leftarrow \mathsf{FS.Sign}(\mathsf{sk}_{f_b}, m_b)$ to the adversary.
- The adversary outputs a bit b', and wins the game if $b' = b$.

Succinctness:
There exists a polynomial $s(\cdot)$ such that for every $k \in \mathbb{N}, f \in \mathcal{F}, m \in \mathcal{D}_f$, it holds with probability 1 over $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{FS.Setup}(1^k); \mathsf{sk}_f \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f); (f(m), \sigma) \leftarrow \mathsf{FS.Sign}(f, \mathsf{sk}_f, m)$ that the resulting signature on $f(m)$ has size $|\sigma| \leq s(k, |f(m)|)$. In particular, the signature size is independent of the size $|m|$ of the input to the function, and of the size $|f|$ of a description of the function f.

3.2 Construction

In this section, we present a construction of a (succinct) functional signature scheme, based on succinct non-interactive arguments of knowledge (SNARKs).

Theorem 3.2. *Assuming the existence of SNARKs for NP, there exists a succinct, function-private functional signature scheme for the class of polynomial-size circuits.*

We achieve this via two steps. We first give a construction of a weaker functional signature scheme, achieving correctness and unforgeability but not succinctness or function privacy, based on one-way functions. We then show how to use any weak functional signature scheme (satisfying correctness and unforgeability), together with a SNARK system, to obtain a functional signature scheme that is additionally succinct and function-private. In a third construction, we demonstrate that if one does not require the signatures to be succinct (but still demand function privacy), this transformation can be achieved based on non-interactive zero-knowledge arguments of knowledge (NIZKAoKs).

We present these three constructions in the following three subsections.

3.2.1 OWF-based construction

In this section we give a construction of a functional signature scheme from any standard signature scheme (i.e., existentially unforgeable under chosen-message attack). Our constructed functional signature scheme satisfies the unforgeability property given in Definition 3.1, but not function privacy or succinctness. Since standard signature schemes can be based on one-way functions (OWF) [Rom90], this shows that we can also construct functional signature schemes under the assumption that OWFs exist.

The main ideas of the construction are as follows. The master signing and verification keys $(\mathsf{msk}, \mathsf{mvk})$ will simply be a standard key pair for the underlying signature scheme. As part of the signing key for a function f, the signer receives a fresh key pair $(\mathsf{sk}, \mathsf{vk})$ for the underlying signature scheme, together with a signature (with respect to mvk) on the function f together with vk. We can think of this signature as a certificate authenticating that the owner of key vk has received permission to sign messages in the range of f. We describe the construction below.

Let $\mathsf{Sig} = (\mathsf{Sig.Setup}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be a signature scheme that is existentially unforgeable under chosen message attack. We construct a functional signature scheme $\mathsf{FS1} = (\mathsf{FS1.Setup}, \mathsf{FS1.KeyGen}, \mathsf{FS1.Sign}, \mathsf{FS1.Verify})$ as follows:

- $\mathsf{FS1.Setup}(1^k)$:
  - Sample a signing and verification key pair for the standard signature scheme $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Sig.Setup}(1^k)$, and set the master signing key to be msk, and the master verification key to be mvk.
- $\mathsf{FS1.KeyGen}(\mathsf{msk}, f)$:
  - choose a new signing and verification key pair for the underlying signature scheme: $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.Setup}(1^k)$.
  - compute $\sigma_{vk} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f|\mathsf{vk})$, a signature of f concatenated with the new verification key vk.
  - create the certificate $c = (f, \mathsf{vk}, \sigma_{vk})$.
  - output $\mathsf{sk_f} = (\mathsf{sk}, c)$.
- $\mathsf{FS1.Sign}(f, \mathsf{sk_f}, m)$:
  - parse $\mathsf{sk_f}$ as (sk, c), where sk is a signing key for the underlying signature scheme, and c is a certificate as described in the KeyGen algorithm.
  - sign m using sk: $\sigma_m \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m)$.
  - output $(f(m), \sigma)$, where $\sigma = (m, c, \sigma_m)$
- $\mathsf{FS1.Verify}(\mathsf{mvk}, m^*, \sigma)$:
  - parse $\sigma = (m, c = (f, \mathsf{vk}, \sigma_{vk}), \sigma_m)$ and check that:
    1. $m^* = f(m)$.
    2. $\mathsf{Sig.Verify}(\mathsf{vk}, m, \sigma_m) \rightarrow 1$: σ_m is a valid signature of m under the verification key vk.
    3. $\mathsf{Sig.Verify}(\mathsf{mvk}, \mathsf{vk}|f, \sigma_{vk}) = 1$: σ_{vk} is a valid signature of $f|\mathsf{vk}$ under the verification key mvk.

Theorem 3.3. *If the signature scheme* Sig *is existentially unforgeable under chosen message attack,* $\mathsf{FS1}$ *as specified above satisfies the unforgeability requirement for functional signatures.*

Proof. Fix a PPT adversary $\mathsf{A_{FS}}$, and let $Q(k)$ be a polynomial upper bound on the number of queries made by $\mathsf{A_{FS}}$ to the oracles $\mathsf{O_{key}}$ and $\mathsf{O_{sign}}$. We will use $\mathcal{A}_{\mathsf{FS}}$ to construct an adversary $\mathsf{A_{sig}}$ such that, if $\mathcal{A}_{\mathsf{FS}}$ wins in the unforgeability game for functional signatures with non-negligible probability, then $\mathsf{A_{sig}}$ breaks the underlying signature scheme, which is assumed to be secure against chosen message attack.

For $\mathsf{A_{FS}}$ to win the functional signature unforgeability game, it must produce a message signature pair (m^*, σ), where $\sigma = (m, (f, \mathsf{vk}, \sigma_{vk}), \sigma_m)$ such that:

- σ_m is a valid signature of m under the verification key vk.
- σ_{vk} is a valid signature of $f|\mathsf{vk}$ under mvk.
- $f(m) = m^*$.
- $\mathsf{A_{FS}}$ has not sent a query of the form $\mathsf{O_{key}}(\tilde{f}, i)$ to the signing key generation oracle for any $\tilde{f}$ that has m^* in its range.
- $\mathsf{A_{FS}}$ hasn't sent a query of the form $\mathsf{O_{sign}}(\tilde{f}, i, \tilde{m})$ to the signing oracle for any $\tilde{f}, \tilde{m}$ such that $\tilde{f}(\tilde{m}) = m^*$

There are two cases for such a forgery (m^*, σ), where $\sigma = (m, (f, \mathsf{vk}, \sigma_{vk}), \sigma_m)$:

- **Type I forgery**: The values (f, vk) are such that the concatenated pair $f|\mathsf{vk}$ has *not* already been signed under mvk during any point of the signing and key oracle queries during the security game.
- **Type II forgery**: The values (f, vk) are such that the concatenated pair $f|\mathsf{vk}$ *has* been signed under mvk during the course of $\mathcal{A}_{\mathsf{FS}}$'s oracle queries.

Here we refer to all mvk signatures generated by the oracles $\mathsf{O_{sign}}$, $\mathsf{O_{key}}$ as intermediate steps in order to answer $\mathsf{A_{FS}}$'s respective queries.

We now describe the constructed signature adversary, $\mathsf{A_{sig}}$. In the security game for the standard (existentially unforgeable under chosen message attack) signature scheme, $\mathsf{A_{sig}}$ is given the verification key $\mathsf{vk_{sig}}$,

and access to a signing oracle $\mathsf{O}_{\mathsf{Reg}_{\mathsf{sig}}}$. He is considered to be successful in producing a forgery if he outputs a valid signature for a message that was not queried from $\mathsf{O}_{\mathsf{Reg}_{\mathsf{sig}}}$.

$\mathsf{A}_{\mathsf{sig}}$ interacts with A_{FS}, playing the role of the challenger in the security game for the functional signature scheme. This means that $\mathsf{A}_{\mathsf{sig}}$ must simulate the $\mathsf{O}_{\mathsf{key}}$ and $\mathsf{O}_{\mathsf{sign}}$ oracles. A_{FS} flips a coin b, indicating his guess for the type of forgery A_{FS} will produce, and places his challenge accordingly.

Case 1: b = 1: $\mathsf{A}_{\mathsf{sig}}$ guesses that A_{FS} will produce a **Type I** forgery:

First $\mathsf{A}_{\mathsf{sig}}$ forwards his challenge verification key $\mathsf{vk}_{\mathsf{sig}}$ to A_{FS} as the master verification key in the functional signature security game.

To simulate the $\mathsf{O}_{\mathsf{key}}$, and $\mathsf{O}_{\mathsf{sign}}$ oracles, $\mathsf{A}_{\mathsf{sig}}$ maintains a dictionary indexed by tuples (f, i), whose entries are signing keys for the functional signature scheme that have already been generated. $\mathsf{A}_{\mathsf{sig}}$ answers the queries issued by A_{FS} as follows:

- $\mathsf{O}_{\mathsf{key}}(f, i)$:
  - if there exists an entry for the key (f, i) in the dictionary, then output the corresponding value, sk_f^i.
  - otherwise, $\mathsf{A}_{\mathsf{sig}}$ generates a new key pair for the underlying signature scheme, $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.Setup}(1^k)$, obtains $\sigma_{vk} \leftarrow \mathsf{O}_{\mathsf{Reg}_{\mathsf{sig}}}(f|\mathsf{vk})$ from its own signing oracle (in the standard signature challenge), and returns $\mathsf{sk_f} = (\mathsf{sk}, \sigma_{vk})$ to A_{FS}. It also sets entry (f, i) in its dictionary to $\mathsf{sk_f}$.
- $\mathsf{O}_{\mathsf{sign}}(f, i, m)$:
  - if there exists an entry for the key (f, i) in the dictionary, $\mathsf{sk}_f^i = (sk, \sigma_{vk})$. It then generates a signature using sk_f^i: that is, generate a signature $\sigma_m \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m)$, and output $(f(m), \sigma)$, where $\sigma = (m, c = (f, \mathsf{vk}, \sigma_{vk}), \sigma_m)$.
  - otherwise, $\mathsf{A}_{\mathsf{sig}}$ generates a new key pair for the regular signature scheme, $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.Setup}(1^k)$, obtains $\sigma_{vk} \leftarrow \mathsf{O}_{\mathsf{Reg}_{\mathsf{sig}}}(f|\mathsf{vk})$ from its signing oracle, and sets entry (f, i) in its dictionary to $\mathsf{sk_f} = (\mathsf{sk}, \sigma_{vk})$. It then generates $\sigma_m \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m)$, and outputs $(f(m), \sigma)$, where $\sigma = (m, c = (f, \mathsf{vk}, \sigma_m), \sigma_{vk})$.

Eventually, A_{FS} outputs a signature (m^*, σ), where $\sigma = (m, (f, \mathsf{vk}, \sigma_{vk}), \sigma_m)$. A_{sig} outputs $(f|\mathsf{vk}, \sigma_{vk})$ as its message-forgery pair in the security game for the standard signature scheme.

Case 2: b = 0: $\mathsf{A}_{\mathsf{sig}}$ guesses that A_{FS} will produce a **Type II** forgery:
$\mathsf{A}_{\mathsf{sig}}$ generates a new key pair $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Sig.Setup}(1^k)$ himself, and forwards mvk to A_{FS}. He also guesses a random index q between 1 and $Q(k)$, denoting which of A_{FS}'s signing queries he will embed his challenge verification key in. He keeps track of the number of keys generated so far in a variable $NUMKEYS$, which is initialized to 0. As before, $\mathsf{A}_{\mathsf{sig}}$ maintains a dictionary indexed by tuples (f, i), whose entries are signing keys for the functional signature scheme that have already been generated. $\mathsf{A}_{\mathsf{sig}}$ answers the queries issued by A_{FS} as follows:

- $\mathsf{O}_{\mathsf{key}}(f, i)$:
  - if there exists an entry for the key (f, i) in the dictionary, with value $CHALLENGE$, abort
  - if there exists an entry for the key (f, i) in the dictionary and its value is not $CHALLENGE$, then output the corresponding value, sk_f^i.
  - otherwise, $\mathsf{A}_{\mathsf{sig}}$ generates a new key pair for the regular signature scheme, $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.Setup}(1^k)$, generates $\sigma_{vk} \leftarrow \mathsf{Sign}(\mathsf{msk}, f|\mathsf{vk})$ himself, and returns $\mathsf{sk_f} = (\mathsf{sk}, \sigma_{vk})$ to A_{FS}. It also sets entry (f, i) in its dictionary to $\mathsf{sk_f}$.
- $\mathsf{O}_{\mathsf{sign}}(f, i, m)$:
  - if there exists an entry for the key (f, i) in the dictionary, $\mathsf{sk}_f^i = (sk, \sigma_{vk})$, generate $\sigma_m \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m)$, and output $(f(m), \sigma)$, where $\sigma = (m, c = (f, \mathsf{vk}, \sigma_{vk}), \sigma_m)$.
  - if there is no (f, i) entry in the dictionary, and $NUMKEYS \neq q$, $\mathsf{A}_{\mathsf{sig}}$ generates a new key pair for the regular signature scheme, $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.Setup}(1^k)$, signs $f|\mathsf{vk}$ himself with respect

to msk: $\sigma_{vk} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f|\mathsf{vk})$, and sets entry (f, i) in its dictionary to $\mathsf{sk_f}$. It then generates a signature on m with respect to the new key sk: $\sigma_m \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m)$, and outputs $(f(m), \sigma)$, where $\sigma = (m, c = (f, \mathsf{vk}, \sigma_{vk}), \sigma_m)$. $NUMKEYS$ is then incremented.

- if there is no (f, i) entry in the dictionary and $NUMKEYS = q$, or if the (f, i) entry in the dictionary is set to $CHALLENGE$, then $\mathsf{A_{sig}}$ queries its oracle for a signature of m under $\mathsf{vk_{sig}}$, $\sigma_m \leftarrow \mathsf{O}_{\mathsf{Reg_{sig}}}(m)$, computes $\sigma_{vk} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f|\mathsf{vk}_{sig})$, and outputs $(f(m), \sigma)$, where $\sigma = (m, c = (f, \mathsf{vk}, \sigma_{vk}), \sigma_m)$. If there is no (f, i) entry in the dictionary, $\mathsf{A_{sig}}$ sets it to $CHALLENGE$. $NUMKEYS$ is then incremented.

If $\mathsf{A_{sig}}$ does not abort, $\mathsf{A_{FS}}$ will eventually output a signature (m^*, σ), where $\sigma = (m, (f, \mathsf{vk}, \sigma_{vk}), \sigma_m)$. $\mathsf{A_{sig}}$ outputs (m, σ_m) as its forgery in the security game for the standard signature scheme with respect to $\mathsf{vk_{sig}}$.

We will now argue that if $\mathsf{A_{FS}}$ forges in the functional signature scheme with non-negligible probability then $\mathsf{A_{sig}}$ is wins the unforgeability game for the standard signature scheme with non-negligible probability.

First note that as long as $\mathsf{A_{sig}}$ does not abort (i.e., the bad situation is not encountered where the adversary requests the secret key corresponding to the embedded $\mathsf{vk_{sig}}$ challenge), then his answers to the $\mathsf{A_{FS}}$'s keygen and signing queries are simulated perfectly as in the real world. Further, as long as there is not an abort, the view of $\mathsf{A_{FS}}$ is independent of $\mathsf{A_{sig}}$'s choice of b and q, as they only determine which verification key is the challenge verification key vk_{sig}

Now, if $\mathsf{A_{FS}}$ produces a **Type I forgery**, then by definition this forgery must include a signature on a new message $f|\mathsf{vk}$ that was *not ever signed* under the master verification key mvk during the course of any oracle query response. Thus, if $\mathsf{A_{FS}}$ makes a Type I forgery and $\mathsf{A_{sig}}$ guessed $b = 1$ (embedding his challenge signature key in the position of the mvk), then $\mathsf{A_{FS}}$'s forgery includes a signature on a new message $f|\mathsf{vk}$ that $\mathsf{A_{sig}}$ did not query to his signature oracle, constituting a forger in the unforgeability game for the standard signature scheme.

If $\mathsf{A_{FS}}$ produces a **Type II forgery**, then the corresponding $f|\mathsf{vk}$ was already signed under the master verification key mvk during the course of one of the oracle queries. This cannot have occurred during a $\mathsf{O_{key}}$ query, as it would mean that $\mathsf{A_{FS}}$ queried $\mathsf{O_{key}}$ on the function f, and producing a signature with respect to this f is not a valid forgery in the functional signature scheme. It must then have been signed during an $\mathsf{O_{sign}}$ query. Namely, the verification key vk must have been freshly generated during a query of the form $\mathsf{O_{sign}}(f, i, m)$ for which no entry under index (f, i) previously existed, and then the pair $f|\mathsf{vk}$ was signed.

Note that if $\mathsf{A_{FS}}$ produces a Type II forgery and $\mathsf{A_{sig}}$ guessed $b = 0$ and the correct q to embed his challenge, and $\mathsf{A_{sig}}$ does not abort, the forgery produced by $\mathsf{A_{FS}}$ must include a signature of a new message $\tilde{m}$ with respect to $\mathsf{vk_{sig}}$, for a $\tilde{m}$ that $\mathsf{A_{sig}}$ hasn't queried from his signing oracle, and therefore $\mathsf{A_{sig}}$ can use this forgery as its own forged signature in the unforgeability game for the standard signature scheme.

We note that, if $\mathsf{A_{sig}}$ does abort, it must be that he embedded his challenge in a query q of the form $\mathsf{O_{sign}}(f, i, m)$, and later $\mathsf{A_{FS}}$ issued a key generation query $\mathsf{O_{sign}}(f, i)$. But this query can't be the signing query q^* for which the adversary receives a signature of $f|\mathsf{vk}$ under mvk, and later outputs a signature of $f(m')$ for another m'. Since the adversary has queried the $\mathsf{O_{sign}}(f, i)$, no message in the range of f would be considered a forgery in the functional signature game. We can conclude that, if $\mathsf{A_{sig}}$ aborts, he didn't guess q^* correctly, so we don't need to consider this case separately.

Denoting by b, q the guesses of $\mathsf{A_{sig}}$, we have that success probability of if $\mathsf{A_{sig}}$ is therefore:

$$
\begin{aligned}
&\Pr[\mathsf{A_{sig}} \text{ forges in signature challenge}] \\
&\quad \geq \Pr[\ b = 1 \ \wedge \ \mathsf{A_{FS}} \text{ outputs Type I forgery }] \\
&\qquad + \sum_{q* \in [Q(k)]} \Pr[\ b = 0 \ \wedge \ q = q* \ \wedge \mathsf{A_{sig}} \text{ does not abort } \wedge \mathsf{A_{FS}} \text{ outputs Type II forgery wrt } \mathsf{vk}_{q*}] \\
&\quad = \Pr[\ b = 1 \ \wedge \mathsf{A_{FS}} \text{ outputs Type I forgery }] \\
&\qquad + \sum_{q* \in [Q(k)]} \Pr[\ b = 0 \ \wedge \ q = q^* \ \wedge \ \mathsf{A_{FS}} \text{ outputs Type II forgery on } vk_{q*}] \\
&\quad \geq \frac{1}{2} \Pr[\mathsf{A_{FS}} \text{ outputs Type I forgery }] + \frac{1}{2Q(k)} \sum_{i* \in [Q(k)]} \Pr[\mathsf{A_{FS}} \text{ outputs Type II forgery on } \mathsf{vk}_{q*}] \\
&\quad \geq \frac{1}{2Q(k)} \left[\Pr[\mathsf{A_{FS}} \text{ outputs Type I forgery }] + \sum_{i* \in [Q(k)]} \Pr[\mathsf{A_{FS}} \text{ outputs Type II forgery on } \mathsf{vk}_{q*}] \right] \\
&\quad = \frac{1}{2Q(k)} \Pr[\mathsf{A_{FS}} \text{ forges}]
\end{aligned}
$$

Thus, if $\mathsf{A_{FS}}$ produces a forgery in the functional signature scheme with non-negligible probability $1/P(k)$, then $\mathsf{A_{sig}}$ successfully forges in the underlying signature scheme with non-negligible probability $1/2P(k)Q(k)$. But, this cannot be the case, since we've assumed that Sig is existentially unforgeable against chosen-message attack. We conclude that $\mathsf{FS1}$ satisfies the unforgeability requirement for functional signatures.

□

While this construction is secure under a very general assumption (the existence of one-way functions), it does not provide function privacy guarantees (indeed, the signature *contains* a description of the relevant pre image and function), and its efficiency can be greatly improved. The size of a signature generated with key sk_f ($\sigma \leftarrow \mathsf{FS.Sign}(sk_f, m)$) in this scheme is proportional to the size of $|f| + |m|$ plus the size of a signature of the standard signature scheme. In contrast, we will next show how to use SNARKs to construct a functional signature where the signature size is proportional to $|f(m)|$, instead of $|f| + |m|$.

3.2.2 Succinct, Function-Private Functional Signatures from SNARKs

We demonstrate how to combine any unforgeable functional signature scheme (such as the OWF-based construction from the previous section) together with a succinct non-interactive argument of knowledge (SNARK) to obtain a new functional signature scheme also satisfying *succinctness and function privacy.*

Let $\mathsf{FS1} = (\mathsf{FS1.Setup}, \mathsf{FS1.Sign}, \mathsf{FS1.Verify})$ be a functional signature scheme, satisfying the unforgeability game as in Definition 3.1, but not necessarily function privacy or succinctness. Let $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}), \mathsf{E} = (\mathsf{E}_1, \mathsf{E}_2))$ be an efficient adaptive zero-knowledge SNARK system for the following NP language L:

$$L = \{(m, \mathsf{mvk}) \mid \exists \sigma \text{ s.t. } \mathsf{FS1.Verify}(\mathsf{mvk}, m, \sigma) = 1\}.$$

We show how to use $\mathsf{FS1}$ and Π to construct a new functional signature scheme that also satisfies function privacy and succinctness.

- $\mathsf{FS2.Setup}(1^k)$:
  - choose a master signing key, verification key pair for $\mathsf{FS1}$: $(\mathsf{msk}', \mathsf{mvk}') \leftarrow \mathsf{FS1.Setup}(1^k)$.
  - choose a crs for the zero-knowledge SNARK: $\mathsf{crs} \leftarrow \Pi.\mathsf{Gen}(1^k)$.
  - set the master secret key $\mathsf{msk} = \mathsf{msk}'$, and the master verification key $\mathsf{mvk} = (\mathsf{mvk}', \mathsf{crs})$.
- $\mathsf{FS2.KeyGen}(\mathsf{msk}, f)$:

– the key generation algorithm is the same as in the underlying functional signature scheme: $\mathsf{sk}_f \leftarrow \mathsf{FS1.KeyGen}(\mathsf{msk}, f)$.

- $\mathsf{FS2.Sign}(f, \mathsf{sk}_f, m)$:
  – generate a signature on m in the underlying functional signature scheme: $\sigma' \leftarrow \mathsf{FS1.Sign}(f, \mathsf{sk}_f, m)$.
  – generate $\pi \leftarrow \Pi.\mathsf{Prove}((f(m), \mathsf{mvk}'), \sigma', \mathsf{crs})$, a zero-knowledge SNARK that $(f(m), \mathsf{mvk}') \in L$, where L is defined as above, and output $(m^* = f(m), \sigma = \pi)$. Informally, π is a proof that the signer knows a signature of $f(m)$ in the underlying functional signature scheme.
- $\mathsf{FS2.Verify}(\mathsf{mvk}, m^*, \sigma)$:
  – output $\Pi.\mathsf{Verify}(crs, m^*, \sigma)$: i.e., verify that σ is a valid argument of knowledge of a signature of $f(m)$ in the underlying functional signature scheme.

Theorem 3.4. *Assume the existence of an unforgeable (but not necessarily succinct or function-private) functional signature scheme* FS1 *supporting the class $\mathcal{F}$ of polynomial-sized circuits, and* Π *be an adaptive zero-knowledge SNARK system for NP. Then there exists* succinct, function-private *functional signatures for $\mathcal{F}$.*

Proof of unforgeability

Suppose there exists an adversary $\mathsf{A_{FS2}}$ that produces a forgery in the new functional signature scheme with non-negligible probability. We show how to construct an adversary $\mathsf{A_{FS1}}$ that uses $\mathsf{A_{FS2}}$ to produce a forgery in the underlying functional signature scheme.

$\mathsf{A_{FS1}}$ plays the role of the challenger in the security game for $\mathsf{A_{FS2}}$. He gets a verification key mvk_{FS1} in his own unforgeability game, generates $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^k)$, a *simulated* CRS for the ZK-SNARK, together with a trapdoor, and forwards $\mathsf{mvk_{FS2}} = (\mathsf{mvk_{FS1}}, \mathsf{crs})$ to $\mathsf{A_{FS2}}$ as the new master verification key. $\mathsf{A_{FS2}}$ makes two types of queries:

- $\mathsf{O_{keyFS2}}(f, i)$, which $\mathsf{A_{FS1}}$ answers (honestly) by forwarding them to its KeyGen oracle, $\mathsf{O_{keyFS1}}(f, i)$
- $\mathsf{O_{signFS2}}(f, m, i)$, in which case $\mathsf{A_{FS1}}$ forwards the query to his signing oracle, and receives a signature $\sigma_{FS1} \leftarrow \mathsf{O_{signFS1}}(f, m, i)$. It then outputs $\pi \leftarrow \Pi.\mathsf{Prove}((f(m), \mathsf{mvk_{FS1}}), \sigma, \mathsf{crs})$ as his signature of $f(m)$.

After querying the oracles, $\mathsf{A_{FS2}}$ will output an alleged forgery in the functional signature scheme, π^*, on some message m^*. $\mathsf{A_{FS1}}$ runs the extractor $\mathsf{E}_2(\mathsf{crs}, \mathsf{trap}, (m^*, \mathsf{mvk_{FS1}}), \pi^*)$ to recover a witness $w = \sigma$ such that $\mathsf{FS1.Verify}(\mathsf{mvk_{FS1}}, m^*, \sigma) = 1$ $\mathsf{A_{sig}}$ then submits σ as a forgery in his own unforgeability game.

We now prove that if $\mathsf{A_{FS2}}$ forges with noticeable probability, then $\mathsf{A_{FS1}}$ also forges with noticeable probability in his own security game.

Hybrid 0. The real-world functional signature challenge experiment. Namely, the CRS is generated in the honest fashion $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$, and the adversary's signing queries are answered honestly. Denote the probability of the adversary producing a valid forgery in the functional signature FS2 scheme within this experiment by Forge_0.

Hybrid 1. The same experiment as Hybrid 0, except the CRS is generated using the extraction-enabling procedure, $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^k)$. The remainder of the experiment continues as before with respect to crs. Denote the probability of the adversary producing a valid forgery in the functional signature scheme within this experiment by Forge_1.

Hybrid 2. The interaction with the adversary is the same as in Hybrid 1. Denote by M the set of all messages signed with $\mathsf{msk_{FS1}}$ in the underlying functional signature scheme during the course of the experiment, as a result of $\mathsf{A_{FS1}}$'s key and signing oracle queries. At the experiment conclusion, the ZK-SNARK extraction algorithm is executed on the adversary's alleged forgery π^* (on message m^*) in the functional signature scheme: i.e., $(\sigma^*) \leftarrow \mathsf{E}_2(\mathsf{crs}, \mathsf{trap}, (m^*, \mathsf{mvk_{FS1}}), \pi^*)$.

Denote by $\mathsf{Extract}_2$ the probability that σ^* is a valid signature in the underlying functional signature scheme FS1 on a message m^* such that $f^* \notin M$. Note that this corresponds to the probability of $\mathsf{A_{FS1}}$ successfully producing a forgery.

Unforgeability of the functional signature scheme follows from the following sequence of lemmas.

Lemma 3.5. $\mathsf{Forge}_0 \leq \mathsf{Forge}_1 + negl(k)$.

Proof. Follows directly from the fact that the CRS values generated via the standard algorithm Gen and those generated by the extraction-enabling algorithm E_1 are *statistically* close, as per Definition 2.8.

More formally, suppose there exists a PPT adversary $\mathcal{A}$ for which $\mathsf{Forge}_1 < \mathsf{Forge}_0 - \epsilon$ for some ϵ. Then the following (not necessarily efficient) adversary $\mathcal{A}_{\mathsf{crs}}$ distinguishes between CRS values with advantage ϵ. In the CRS challenge, $\mathcal{A}_{\mathsf{crs}}$ is given a value crs (generated by either the standard algorithm or the extraction-enabling algorithm). First, $\mathcal{A}_{\mathsf{crs}}$ generates a key pair $(\mathsf{msk}_{FS1}, \mathsf{mvk}_{FS1}) \leftarrow \mathsf{FS1.Setup}(1^k)$ for the underlying functional signature scheme, and sends $\mathsf{mvk}_{FS2} = (\mathsf{mvk}_{FS1}, \mathsf{crs})$ to $\mathcal{A}$. He answers $\mathcal{A}_{FS2}$'s queries as in Hybrid 0, generating signatures and proofs as required (note that $\mathcal{A}$ holds the master secret key msk_{FS1}, which allows him to answer the queries). At the conclusion of $\mathcal{A}_{FS2}$'s query phase, he outputs an alleged forgery π^* in the functional signature scheme. The adversary $\mathcal{A}_{\mathsf{crs}}$ tests whether π^* is indeed a forgery. We note that this verification process might not be efficient, since $\mathcal{A}_{\mathsf{crs}}$ needs to test whether the message whose signature A_{FS2} claims to have forged is actually not in the range of any of the functions f that $\mathcal{A}_{FS2}$ has requested signing keys for. If the forgery verifies, $\mathcal{A}_{\mathsf{crs}}$ outputs "standard crs"; otherwise, he outputs "extractable crs". His advantage in the CRS distinguishing game is precisely $\mathsf{Forge}_1 - \mathsf{Forge}_0$, as desired. Since the real and simulated CRS strings are supposed to be statistically close, the distinguishing advantage $\mathsf{Forge}_1 - \mathsf{Forge}_0$ has to be negligible even for an inefficient adversary. □

Lemma 3.6. $\mathsf{Forge}_1 \leq \mathsf{Extract}_2 + negl(k)$.

Proof. This holds by the extraction property of the ZK-SNARK system (Definition 2.8).

Namely, if there exists a PPT adversary $\mathcal{A}$ for which $\mathsf{Forge}_1 > \mathsf{Extract}_2 + \epsilon$ for some ϵ, then the following adversary $\mathcal{A}_{\mathsf{Ext}}$ successfully produces a properly-verifying proof π for which extraction fails with probability ϵ (which must be negligible by the SNARK extraction property).

$\mathcal{A}_{\mathsf{Ext}}$ receives a CRS value crs generated via $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^k)$. He samples a key pair $(\mathsf{msk}_{FS1}, \mathsf{mvk}_{FS1})$ $\leftarrow \mathsf{FS1.Setup}(1^k)$ for the underlying functional signature scheme, sends $\mathsf{mvk}_{FS2} = (\mathsf{mvk}_{FS1}, \mathsf{crs})$ to the adversary $\mathcal{A}$, and answers all of $\mathcal{A}$'s key and signing oracle queries as in Hybrid 1.

Now, let M the collection of all messages f which were signed by $\mathcal{A}_{\mathsf{Ext}}$ during the course of the interaction with $\mathcal{A}$. Suppose that π^* is a valid forgery on m^* in the functional signature scheme; in particular, π^* is a valid proof that $(m^*, \mathsf{mvk}_{FS1}) \in L$. We argue that if extraction succeeds on π^* (i.e if $\sigma^* \leftarrow \mathsf{E}_2(\mathsf{crs}, \mathsf{trap}, (m^*, \mathsf{vk}), \pi^*)$ yields a valid witness for $(m^*, \mathsf{vk}) \in L)$), then it must be that the extracted σ^* is a valid signature on a message $g \notin M$ in the underlying functional signature scheme, so that we are in the event corresponding to $\mathsf{Extract}_3$. That is, we show $\mathsf{Forge}_1 - \mathsf{Extract}_2$ is bounded above by the probability that extraction *fails*.

Since π^* is a valid forgery in the functional signature scheme $\mathsf{FS2}$, it must be that $m^* \notin Range(g)$ for all key queries $\mathsf{O}_{\mathsf{key}}(g, i)$ made by $\mathcal{A}$, and that $m^* \neq g(x)$ for all signing queries $\mathsf{O}_{\mathsf{sign}}(g, x, i)$ made by $\mathcal{A}$. Now, if the extracted tuple $(f^*, m, \sigma^*) \leftarrow \mathsf{E}_2(\mathsf{crs}, \mathsf{trap}, (m^*, \mathsf{vk}), \pi^*)$ is a valid witness for $(m^*, \mathsf{vk}) \in L$, then from the definition of the language L it means that $m^* = f^*(m)$ and that σ^* is a valid signature on f^* with respect to the master signing key sk (i.e., $\mathsf{Verify}(\mathsf{vk}, \sigma^*, f^*) = 1$). Recall that the set M consists exactly of the functions g for which $\mathcal{A}$ made a key query, and the collection of *constant functions* $g' \equiv g(x)$ for which $\mathcal{A}$ make a signing query (g, x). But since $m^* \in Range(f^*)$ and $m^* \notin Range(g)$ for all $g \in M$, it must be that $f^* \notin M$, as desired.

Therefore, with probability at least $\mathsf{Forge}_2 - \mathsf{Extract}_3 = \epsilon$, it must hold that π^* is a valid proof but that the extraction algorithm *fails* to extract a valid witness from π^*. By the extraction property of the SNARK system, it must be that ϵ is negligible.

□

Lemma 3.7. $\mathsf{Extract}_2 < negl(k)$.

Proof. This holds by the unforgeability of the underlying functional signature scheme $\mathsf{FS1}$, since $\mathsf{Extract}_2$ is precisely the probability that adversary $\mathsf{A}_{\mathsf{FS1}}$ constructed above produces a successful forgery in the unforgeability game for $\mathsf{FS1}$. □

Proof of function privacy
We show that any adversary $\mathsf{A}_{\mathsf{priv}}$ who succeeds in the function privacy game with noticeable advantage can be used to break the zero knowledge property of the ZK-SNARK scheme. Recall that in the adaptive zero knowledge security game, the adversary is given a CRS (either honestly generated or simulated) and access to an oracle who accepts statement-witness pairs (x, w) and responds with either honestly generated or simulated proofs of the statement.

More specifically, consider the following two hybrid experiments:

Hybrid 0. The real function privacy challenge. In particular, the CRS for the ZK-SNARK system is generated honestly as $\mathsf{crs} \leftarrow \Pi.\mathsf{Gen}(1^k)$. The challenge signature, on message m_b for randomly chosen $b \leftarrow \{0,1\}$ (with respect to key $\mathsf{f_b}$), is generated by first generating a signature on m_b in the underlying functional signature scheme $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_{f_b}, m_b)$ and then honestly generating a proof $\pi \leftarrow \Pi.\mathsf{Prove}((f_b(m_b), \mathsf{mvk}), \sigma, \mathsf{crs})$.

Hybrid 1. Similar to Hybrid 0, except that the SNARK appearing in the challenge signature is replaced by a *simulated* argument. Namely, the CRS is generated using the simulator algorithm $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)$. And the challenge signature is generated by sampling a random bit $b \leftarrow \{0,1\}$ and *ignoring* it, instead using the simulator $\pi \leftarrow \mathcal{S}^{\mathsf{Proof}}(\mathsf{crs}, \mathsf{trap}, (m', \mathsf{mvk}))$, where $m' = f_0(m_0) = f_1(m_1)$.

Denote by $\mathsf{win}_0, \mathsf{win}_1$ the advantage of the adversary $\mathsf{A}_{\mathsf{priv}}$ in guessing the bit b in Hybrid 0 and 1, respectively. Function privacy of $\mathsf{FS2}$ follows from the following two claims.

Claim 3.8. $\mathsf{win}_1 \geq \mathsf{win}_0 - \mathsf{negl}(k)$.

Proof. Follows directly from the adaptive zero knowledge property of the ZK-SNARK system. More explicitly, consider the following adversary A_{ZK}:

1. A_{ZK} receives a CRS value crs from the adaptive zero knowledge challenger (either honestly generated or simulated). In addition, he generates a master key pair for the underlying functional signature scheme: $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{FS1.Setup}(1^k)$. A_{ZK} takes $\mathsf{mvk}' = (\mathsf{mvk}, \mathsf{crs})$ and sends the key pair $(\mathsf{mvk}', \mathsf{msk})$ to the function privacy adversary $\mathsf{A}_{\mathsf{priv}}$.
2. $\mathsf{A}_{\mathsf{priv}}$ responds (adaptively) with function queries f_0, f_1 and a message pair m_0, m_1 with $f_0(m_0) = f_1(m_1)$. For each function query f_b, A_{ZK} generates a corresponding key $\mathsf{sk}_{f_b} \leftarrow \mathsf{FS1.KeyGen}(\mathsf{msk}, f_b)$ and sends sk_{f_b} to $\mathsf{A}_{\mathsf{priv}}$.
3. A_{ZK} prepares the function privacy challenge signature as follows. First, he chooses a random bit $b \leftarrow \{0,1\}$, and uses $(f_b, m_b, \mathsf{sk}_{f_b})$ to generate a signature on $f_b(m_b)$ in the underlying functional signature scheme: $\sigma \leftarrow \mathsf{FS1.Sign}(\mathsf{sk}_{f_b}, m_b)$. He then submits the query $((f_b(m_b), \mathsf{mvk}), \sigma)$ to the proof oracle in his own ZK challenge. (Recall that σ is a valid witness for $(f_b(m_b), \mathsf{mvk}) \in L$). Denote the oracle response by π, which is either honestly generated or simulated.
4. A_{ZK} sends the signature π to $\mathsf{A}_{\mathsf{priv}}$, who responds with a guessed bit b' in the function privacy game. If $b' = b$, then A_{ZK} outputs "real." Otherwise, if $b' \neq b$, then A_{ZK} outputs "simulated."

Note that if A_{ZK} has access to the Real Proof experiment (Experiment $\mathsf{Exp}_{\mathcal{A}}(k)$ in Definition 2.9), then A_{ZK} perfectly simulates Hybrid 0, whereas if he has access to the Simulated Proof experiment (Experiment $\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}(k)$ in Definition 2.9), then A_{ZK} perfectly simulates Hybrid 1. Thus, A_{ZK}'s advantage in the adaptive zero knowledge challenge is equal to $\mathsf{win}_0 - \mathsf{win}_1$, which by the ZK security of the ZK-SNARK scheme must hence be negligible.

□

Claim 3.9. $\mathsf{win}_1 < \mathsf{negl}(k)$.

Proof. Note that the view of $\mathsf{A}_{\mathsf{priv}}$ in Hybrid 1 is in fact *independent* of the selected bit b. Indeed, the challenge signature is generated with respect *only* to the value $m' = f_0(m_0) = f_1(m_1)$, and not any particular witness. Thus, information theoretically, even a computationally unbounded adversary could not correctly guess the bit b with noticeable advantage. □

Succinctness
The succinctness of our signature scheme follows directly from the succinctness property of the SNARK system. Namely, the size of a functional signature produced by $\mathsf{FS2.Sign}(f, \mathsf{sk}_f, m)$ is exactly the proof length of a SNARK for the language L. There exists a polynomial q such that the runtime R of the associated relation is bounded by $q(|f(m)| + |\mathsf{mvk}| + |\sigma|)$, where σ is a signature in the underlaying, non-succinct functional signature scheme.
By Definition 2.7, there exists a polynomial p, such that the corresponding proof length is bounded by $p(k + polylog(|f(m)| + |\mathsf{mvk}| + |\sigma|))$. The size of the signature $|\sigma| = poly(|f| + |m| + k)$. We may assume that $|f|$, and $|m|$ are bounded by 2^k, and therefore the size of a signature in the SNARK-based construction is polynomial in k, and independent of $|f|$, $|m|$, (and even $|f(m)|$).

3.2.3 NIZK-based construction

If one wishes to avoid SNARK-type assumptions, one can obtain a functional signature scheme satisfying both unforgeability and function privacy (but not succinctness) under the more general assumption of standard non-interactive zero-knowledge arguments of knowledge (NIZKAoK). This can be done by essentially replacing the ZK-SNARKs in the construction of the previous section with NIZKAoKs. We remark that our construction hides the function f, but it reveals the size of a circuit computing f.[3]

Let (FS3.Setup, FS3.Keygen, FS3.Sign, FS3.Verify) be a functional signature scheme which is identical to our previous construction FS2, except that we use a NIZKAoK Π', instead of the zero-knowledge SNARK system Π.

Theorem 3.10. *If (*Sig.Setup*,* Sig.Sign*,* Sig.Verify*) is an existentially unforgeable signature scheme, and Π' is a NIZKAoK, our new functional signature construction (*FS3.Setup*,* FS3.Keygen*,* FS3.Sign*,* FS3.Verify*) satisfies both unforgeability and function privacy.*

We can use the proof from the previous section, since a zero-knowledge SNARK and a NIZK satisfy the same adaptive zero-kowledge and extractability properties that are used in the proof. The only difference is that a SNARK has a more efficient verification algorithm, and shorter proofs, while a NIZK can be constructed under more general assumptions.

4 Applications of Functional Signatures

In this section we discuss applications of functional signatures to other cryptographic problems, such as constructing delegation schemes and succinct non-interactive arguments.

4.1 SNARGs from Functional Signatures

Recall that in a SNARG protocol for a language L, there is a verifier V, and a prover P who wishes to convince the verifier that an input x is in L. To achieve succinctness, proofs produced by the prover must be sublinear in the size of the input plus the size of the witness.

We show how to use a functional signature scheme supporting keys for functions f describable as polynomial-size circuits, and which has short signatures (i.e of size $r(k) \cdot (|f(m)| + |m|)^{o(1)}$ for a polynomial $r(\cdot)$) to construct a SNARG scheme with preprocessing for any language $L \in NP$ with proof size bounded by $r(k) \cdot (|w| + |x|)^{o(1)}$, where w is the witness and x is the instance. We note that this is the proof size used in the lower bound of [GW11].

Let L be an NP complete language, and R the corresponding relation. The main idea in the construction is for the verifier (or CRS setup) to give out a single signing key for a function whose range consists of exactly those strings that are in L. Note that this can be efficiently described by use of the relation R (where the function also takes as input a witness). Then, with sk_f for this appropriate function f, the prover will

[3]This is not a concern in the SNARK-base construction, since the size of the signature was independent of the function size.

be able to sign *only* those messages that are in the language L, and hence can use a signature on x as a convincing proof that $x \in L$. The resulting proof is succinct and publicly verifiable.

More explicitly, let $\mathsf{FS} = (\mathsf{FS.Setup}, \mathsf{FS.KeyGen}, \mathsf{FS.Sign}, \mathsf{FS.Verify})$ be a succinct functional signature scheme (as in Definition 3.1) supporting the class $\mathcal{F}$ of polynomial-size circuits. We construct the desired SNARG system $\Pi = (\Pi.\mathsf{Gen}, \Pi.\mathsf{Prove}, \Pi.\mathsf{Verify})$ for NP language L with relation R, as follows:

- $\Pi.\mathsf{Gen}(1^k)$:
  - run the setup for the functional signature scheme, and get $(\mathsf{mvk}, \mathsf{msk}) \leftarrow \mathsf{FS.Setup}(1^k)$
  - generate a signing key $\mathsf{sk}_f \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f)$ where f is the following function:
    $$f(x|w) := \begin{cases} x \text{ if } R(x,w) = 1 \\ \perp \text{ otherwise} \end{cases} .$$
  - output $\mathsf{crs} = (\mathsf{mvk}, \mathsf{sk}_f)$
- $\Pi.\mathsf{Prove}(x, w, \mathsf{crs})$:
  - output $\mathsf{FS.Sign}(f, \mathsf{sk}_f, x|w)$
- $\Pi.\mathsf{Verify}(\mathsf{crs}, x, \pi)$:
  - output $\mathsf{FS.Verify}(\mathsf{mvk}, x, \pi)$

Theorem 4.1. *If* FS *is a functional signature scheme supporting the class* $\mathcal{F}$ *of polynomial-sized circuits, then* Π *is a succinct non-interactive argument (SNARG) for NP language* L*.*

Proof. We address the correctness, soundness, and succinctness of the scheme.

Correctness
The correctness property of the SNARG scheme follows immediately from correctness property of the functional signature scheme. Namely, let R be the relation corresponding to the language L. Then
$\forall (x, w) \in R, \forall \mathsf{crs} = (\mathsf{mvk}, \mathsf{sk}_f)$, where $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{FS.Setup}(1^k)$, and $\mathsf{sk}_f \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f)$, and $\forall \pi = \sigma$, where $(x, \sigma) \leftarrow \mathsf{FS.Sign}(f, \mathsf{sk}_f, (x, w))$,

$$\Pi.\mathsf{Verify}(\mathsf{crs}, x, \pi) = \mathsf{FS.Verify}(\mathsf{mvk}, x, \sigma) \rightarrow 1.$$

Soundness
The soundness of the proof system follows from the unforgeability property of the signature scheme: since the prover is not given keys for any function except f, he can only sign messages x that are in the range of f, and therefore instances in the language L.
Suppose there exists a PPT adversary Adv for which $\Pr[\mathsf{crs} \leftarrow \Pi.\mathsf{Gen}(1^k); (x, \pi) \leftarrow \mathsf{Adv}(\mathsf{crs}) : x \notin L \wedge \Pi.\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1] = \epsilon(|x|)$, for a non-negligible function $\epsilon(\cdot)$.
Then we can construct an adversary $\mathsf{A_{FS}}$ who breaks the unforgeability of the underlying functional signature scheme. $\mathsf{A_{FS}}$ gives $\mathsf{crs} = (\mathsf{mvk}, \mathsf{sk}_f)$ to Adv, where mvk is his challenge verification key, and sk_f is the signing key for the function f defined above, which he gets from his key generation oracle.
Adv outputs (x, π), and $\mathsf{A_{FS}}$ uses them as his forgery in the functional signature game. If $x \notin L$, x must not be in the range of L, and therefore (x, π) is a valid forgery. So $\mathsf{A_{FS}}$ wins the unforgeability game with probability $\epsilon(|x|)$, which we have assumed is non-negligible.

Succinctness
The size of a proof is equal to the size of a signature in the functional signature scheme, which by assumption is $r(k) \cdot (|f(m)| + |m|)^{o(1)} = r(k) \cdot (|x| + |w|)^{o(1)}$ for a polynomial $r(\cdot)$. □

Remark 4.2 (Functional PRFs as Functional MACs)**.** Note that functional pseudorandom functions directly imply a notion of functional message authentication codes (MACs), where the master PRF seed s serves as the (shared) master secret MAC key, and a functional PRF subkey sk_f enables one to both MAC and verify messages $f(m)$. Using the transformation above with such a functional MAC in the place of functional signatures yields a *privately verifiable* SNARG system.

Remark 4.3 (Lower bound of [GW11])**.** Gentry and Wichs showed in [GW11] that SNARG schemes for NP with proof size $r(k) \cdot (|x| + |w|)^{o(1)}$ for polynomial $r(\cdot)$ cannot be obtained using black-box reductions to falsifiable assumptions [Nao03]. Therefore, combined with Theorem 4.1, it follows that in order to obtain a functional signature scheme with signature size $r(k) \cdot (|f(m)| + |m|)^{o(1)}$ we must either rely on non-falsifiable assumptions (as in our SNARK-based construction) or make use of non black-box techniques.

4.2 Connection between functional signatures and delegation

Recall that a delegation scheme allows a client to outsource the evaluation of a function f to a server, while allowing the client to verify the correctness of the computation. The verification process should be more efficient than computing the function. See Definition 2.12 for the required correctness and security properties.

Given a functional signature scheme with with signature size $\delta(k)$, and verification time $t(k)$ (which we assume is independent of the of the size of a function f used in the signing process), we can get a delegation scheme in the preprocessing model with proof size $\delta(k)$ and verification time $t(k)$. Here k is the security parameter.

Let $(\mathsf{FS.Setup}, \mathsf{FS.Prove}, \mathsf{FS.Sign}, \mathsf{FS.Verify})$ be a functional signature scheme supporting the class $\mathcal{F}$ of polynomial-sized circuits. We construct a delegation scheme $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Compute}, \mathsf{Verify})$ as follows:

- $\mathsf{KeyGen}(1^k, f)$:
  - run the setup for the functional signature scheme and generate $(\mathsf{mvm}, \mathsf{msk}) \leftarrow \mathsf{FS.Setup}(1^k)$.
  - define the function $f'(x) := (x, f(x))$, and generate a signing key for f': $\mathsf{sk}_{f'} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f')$.
  - output $\mathsf{enc} = \perp$, $\mathsf{evk} = \mathsf{sk}_f$, $\mathsf{vk} = \mathsf{mvk}$.
- $\mathsf{Encode}(\mathsf{enc}, x) = x$: no processing needs to be done on the input.
- $\mathsf{Compute}(\mathsf{evk}, \sigma_x)$:
  - let $\mathsf{sk}_{f'} = \mathsf{evk}, x = \sigma_x$
  - generate a signature of $(x, f(x))$ using key $\mathsf{sk}_{f'}$: i.e., $\sigma \leftarrow \mathsf{FS.Sign}(\mathsf{sk}_{f'}, f', x)$
  - output $(f(x), \pi = \sigma)$
- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi_y)$:
  - output $\mathsf{FS.Verify}(\mathsf{vk}, y, \pi_y)$

Theorem 4.4. *If* FS *is a functional signature scheme supporting the class $\mathcal{F}$ of polynomial-sized circuits, then* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Compute}, \mathsf{Verify})$ *is a delegation scheme.*

Correctness
The correctness of the delegation scheme follows from the correctness of the functional signature scheme.

Authenticity
By the unforgeability property of the functional signature scheme, any PPT server will only be able to produce a signature of (x, y) that is in the range of f': that is, if $y = f(x)$. Thus the server will not be able to sign a pair (x, y) with non-negligible probability, unless $y = f(x)$.

Efficiency
The runtime of the verification algorithm of the delegation scheme is the runtime of the verification algorithm for the signature scheme, $t(k)$. The poof size is equal to the size of a signature in the functional signature scheme, $\delta(k)$.

5 Functional Pseudorandom Functions

In this section we present a formal definition of functional pseudorandom functions (F-PRF), pseudorandom functions with selective access (PRF-SA), and hierarchical functional pseudorandom functions. We present a construction of a functional pseudorandom function family supporting the class of *prefix-fixing functions* based on one-way functions, making use of the Goldreich-Goldwasser-Micali (GGM) tree-based PRF construction [GGM86]. Our construction directly yields a PRF with selective access, and additionally supports hierarchical key generation.

5.1 Definition of Functional PRF

In a standard pseudorandom function family, the ability to evaluate the chosen function is all-or-nothing: a party who holds the secret seed s can compute $F_s(x)$ on all inputs x, whereas a party without knowledge of s cannot distinguish evaluations $F_s(x)$ on requested inputs x from random. We propose the notion of a *functional pseudorandom function (F-PRF)* family, which partly fills this gap between evaluation powers. The idea is that, in addition to a master secret key that can be used to evaluate the pseudorandom function F on any point in the domain, there are additional *secret keys per function* f, which allow one to evaluate F on y for any y for which there exists an x such that $f(x) = y$ (i.e., y is in the range of f).

Definition 5.1 (Functional PRF)**.** We say that a PRF family $\mathcal{F} = \{F_s : D \to R\}_{s \in S}$ is a *functional pseudorandom function (F-PRF)* if there exist additional algorithms

$\mathsf{KeyGen}(s, f)$: On input a seed $s \in S$ and function description $f : A \to D$ from some domain A to D, the algorithm KeyGen outputs a key sk_f.

$\mathsf{Eval}(\mathsf{sk}_f, f, x)$: On input key sk_f, function $f : A \to D$, and input $x \in A$, then Eval outputs the PRF evaluation $F_s(f(x))$.

which satisfy the following properties:

- **Correctness:** For every (efficiently computable) function $f : A \to D$, $\forall x \in A$, it holds that

$$\forall s \leftarrow S,\ \forall \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(s, f),\ \mathsf{Eval}(\mathsf{sk}_f, f, x) = F_s(f(x)).$$

- **Pseudorandomness:** Given a set of keys $\mathsf{sk}_{f_1} \dots \mathsf{sk}_{f_l}$ for functions $f_1 \dots f_l$, the evaluation of $F_s(y)$ should remain pseudorandom on all inputs y that are not in the range of any of the functions $f_1 \dots f_l$. That is, for any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in distinguishing between the following two experiments is negligible (for any polynomial $l = l(k)$):

Experiment Rand	Experiment PRand
Key query Phase	Key query Phase
$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$	$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$
$f_1 \leftarrow \mathcal{A}(\mathsf{pp})$	$f_1 \leftarrow \mathcal{A}(\mathsf{pp})$
$\mathsf{sk}_{f_1} \leftarrow \mathsf{KeyGen}(s, f_1)$	$\mathsf{sk}_{f_1} \leftarrow \mathsf{KeyGen}(s, f_1)$
$\vdots$	$\vdots$
$f_l \leftarrow \mathcal{A}(\mathsf{pp}, f_1, sk_{f_1}, \dots, f_{l-1}, sk_{f_{l-1}})$	$f_l \leftarrow \mathcal{A}(\mathsf{pp}, f_1, sk_{f_1}, \dots, f_{l-1}, sk_{f_{l-1}})$
$\mathsf{sk}_{f_l} \leftarrow \mathsf{KeyGen}(s, f_l)$	$\mathsf{sk}_{f_l} \leftarrow \mathsf{KeyGen}(s, f_l)$
Challenge Phase	Challenge Phase
$H \leftarrow \mathbb{F}_{D \to R}$ a random function	
$b \leftarrow \mathcal{A}^{\mathcal{O}^{\{f_i\}}_{s,H}(\cdot)}(f_1, \mathsf{sk}_{f_1}, \dots, f_l, \mathsf{sk}_{f_l})$	$b \leftarrow \mathcal{A}^{F_s(\cdot)}(f_1, \mathsf{sk}_{f_1}, \dots, f_l, \mathsf{sk}_{f_l})$

where $\mathcal{O}^{\{f_i\}}_{s,H}(y) := \begin{cases} F_s(y) & \text{if } \exists i \in [l] \text{ and } x \text{ s.t. } f_i(x) = y \\ H(y) & \text{otherwise} \end{cases}$.

Note that, as defined, the oracle $\mathcal{O}_{s,H}^{\{f_i\}}(y)$ need not be efficiently computable. This inefficiency stems both from sampling a truly random function H, and from testing whether the adversary's evaluation queries y are contained within the range of one of his previously queried functions f_i. However, within particular applications, the system can be set up so that this oracle is efficiently simulatable: For example, evaluations of a truly random function can be simulated by choosing each queried evaluation one at a time; Further, the range of the relevant functions f_i may be efficiently testable given trapdoor information (e.g., determining the range of $f : r \mapsto \mathsf{Enc}(\mathsf{pk}, 0; r)$ for a public-key encryption scheme is infeasible given only pk but efficiently testable given the secret key).

We also consider a weaker security definition, where the adversary has to reveal which functions he will request keys for before seeing the public parameters or any of the keys. We refer to this as *selective pseudorandomness*.

Definition 5.2 (Selectively Secure F-PRF)**.** We say a PRF family is a *selectively secure* functional pseudorandom function if the algorithms $\mathsf{KeyGen}, \mathsf{Eval}$ satisfy the correctness property above, and the following selective pseudorandomness property.

- **Selective Pseudorandomness:** For any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in distinguishing between the following two experiments is negligible:

Experiment Sel-Rand	Experiment Sel-PRand
Key query Phase	Key query Phase
$f_1, \ldots, f_l \leftarrow \mathcal{A}$	$f_1, \ldots, f_l \leftarrow \mathcal{A}$
$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$	$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$
$\mathsf{sk}_{f_1} \ldots \mathsf{sk}_{f_l} \leftarrow \mathsf{KeyGen}(s, f_1, \ldots f_l)$	$\mathsf{sk}_{f_1} \ldots \mathsf{sk}_{f_l} \leftarrow \mathsf{KeyGen}(s, f_1, \ldots f_l)$
Challenge Phase	Challenge Phase
$H \leftarrow \mathbb{F}_{D \to R}$ a random function	
$b \leftarrow \mathcal{A}^{\mathcal{O}_{s,H}^{\{f_i\}}(\cdot)}(f_1, \mathsf{sk}_{f_1}, \ldots f_l, \mathsf{sk}_{f_l})$	$b \leftarrow \mathcal{A}^{F_s(\cdot)}(f_1, \mathsf{sk}_{f_1}, \ldots f_l, \mathsf{sk}_{f_l})$

where $\mathcal{O}_{s,H}^{\{f_i\}}(y) := \begin{cases} F_s(y) & \text{if } \exists i \in [l] \text{ and } x \text{ s. t. } f_i(x) = y \\ H(y) & \text{otherwise} \end{cases}$.

A special case of functional PRFs are when access control is to be determined by predicates. (Indeed, fitting within the F-PRF framework, one can emulate predicate policies by considering the corresponding functions $f_P(x) = x$ if $P(x) = 1$ and $= \perp$ if $P(x) = 0$). For completeness, we now present the corresponding formal definition, which we refer to as PRFs with *selective access*.

Definition 5.3 (PRF with Selective Access)**.** We say that a PRF family $\mathcal{F} = \{F_s : D \to R\}_{s \in S}$ is a *pseudorandom function family with selective access (PRF-SA)* for a class of predicates $\mathcal{P}$ on D if there exist additional efficient algorithms

$\mathsf{KeyGen}(s, P)$: On input a seed $s \in S$ and predicate $P \in \mathcal{P}$, KeyGen outputs a key sk_P.

$\mathsf{Eval}(\mathsf{sk}_P, P, x)$: On input key sk_P and input $x \in D$, if it holds that $P(x) = 1$ then Eval outputs the PRF evaluation $F_s(x)$.

which satisfy the following properties:

- **Correctness:** For each predicate $P \in \mathcal{P}$, $\forall x \in D$ *s.t.* $P(x) = 1$, it holds that

$$\forall s \leftarrow S, \forall \mathsf{sk}_P \leftarrow \mathsf{KeyGen}(s, P), \mathsf{Eval}(\mathsf{sk}_P, P, x) = F_s(x)$$

- **Pseudorandomness:** Given a set of keys $\mathsf{sk}_{P_1} \ldots \mathsf{sk}_{P_l}$ for predicate $P_1 \ldots P_l$, the evaluation of $F_s(x)$ should remain pseudorandom on all inputs x for which $P_1(x) = 0 \wedge \cdots \wedge P_l(x) = 0$. That is, for any PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in distinguishing between the following two experiments is negligible:

Experiment Rand	Experiment PRand
Query Phase	Query Phase
$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$	$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$
$P_1 \leftarrow \mathcal{A}(\mathsf{pp})$	$P_1 \leftarrow \mathcal{A}(\mathsf{pp})$
$\mathsf{sk}_{P_1} \leftarrow \mathsf{KeyGen}(s, P_1)$	$\mathsf{sk}_{P_1} \leftarrow \mathsf{KeyGen}(s, P_1)$
$\vdots$	$\vdots$
$P_l \leftarrow \mathcal{A}(\mathsf{pp}, P_1, sk_{P_1} \dots P_{l-1}, sk_{P_{l-1}})$	$P_l \leftarrow \mathcal{A}(\mathsf{pp}, P_1, sk_{P_1} \dots P_{l-1}, sk_{P_{l-1}})$
$\mathsf{sk}_{P_l} \leftarrow \mathsf{KeyGen}(s, P_l)$	$\mathsf{sk}_{P_l} \leftarrow \mathsf{KeyGen}(s, P_l)$
Challenge Phase	Challenge Phase
$H \leftarrow \mathbb{F}_{D \to R}$ a random function	
$b \leftarrow \mathcal{A}^{\mathcal{O}^P_{s,H}(\cdot)}(P_1, \mathsf{sk}_{P_1}, \dots P_l, \mathsf{sk}_{P_l})$	$b \leftarrow \mathcal{A}^{F_s(\cdot)}(P_1, \mathsf{sk}_{P_1}, \dots P_l, \mathsf{sk}_{P_l})$

where $\mathcal{O}^P_{s,H}(x) := \begin{cases} F_s(x) & \text{if } \exists i \in [l], P_i(x) = 1 \\ H(x) & \text{otherwise} \end{cases}$.

Finally, we consider *hierarchical* F-PRFs, where a party holding key sk_f for function $f : B \to D$ can generate a subsidiary key $\mathsf{sk}_{f \circ g}$ for a second function $g : A \to B$.

Definition 5.4 (Hierarchical F-PRF)**.** We say that an F-PRF family $(\{F_s\}_s, \mathsf{KeyGen}, \mathsf{Eval})$ is *hierarchical* if the algorithm KeyGen is replaced by a more general algorithm:

$\mathsf{SubkeyGen}(\mathsf{sk}_f, g)$: On input a functional secret key sk_f for function $f : B \to C$ (where the master secret key is considered to be sk_1 for the identity function $f(x) = x$), and function description $g : A \to B$ for some domain A, $\mathsf{SubkeyGen}$ outputs a secret subkey $\mathsf{sk}_{f \circ g}$ for the composition $f \circ g$.

satisfying the following properties:

- **Correctness:** Any key sk_g generated via a sequence of $\mathsf{SubkeyGen}$ executions will correctly evaluate $F_s(f(x))$ on each value y for which they know a preimage x with $g(x) = y$. Formally, for every sequence of (efficiently computable) functions $f_1, \dots, f_\ell$ with $f_i : A_i \to A_{i-1}$, $\forall y \in A_0$ s.t. $\exists x \in A_\ell$ for which $f_1 \circ \cdots \circ f_\ell(x) = y$, it holds that

$$\forall \mathsf{sk}_1 \leftarrow S, \quad \forall \mathsf{sk}_{f_1 \circ \cdots \circ f_i} \leftarrow \mathsf{SubkeyGen}(\mathsf{sk}_{f_1 \circ \cdots \circ f_{i-1}}, f_i) \text{ for } i = 0, \dots, \ell,$$
$$\mathsf{Eval}(\mathsf{sk}_{f_1 \circ \cdots \circ f_\ell}, (f_1 \circ \cdots \circ f_\ell), x) = F_{\mathsf{sk}_1}(y).$$

- **Pseudorandomness:** The pseudorandomness property of Definition 5.1 holds, with the slight modification that the adversary may adaptively make queries of the following kind, corresponding to receiving subkeys sk_g generated from unknown functional keys sk_f. The query phase begins with a master secret key $s \leftarrow S$ being sampled and assigned identity $id = 1$. Loosely, $\mathsf{GenerateKey}$ generates a new subkey of an existing (possibly unknown) key indexed by id, and keeps the resulting key hidden. $\mathsf{RevealKey}$ simply reveals the generated key indexed by id.

  $\mathsf{GenerateKey}(id, g)$: If no key exists with identity id then output $\perp$ and terminate; otherwise denote this key by sk_f. The challenger generates a g-subkey from sk_f as $\mathsf{sk}_{f \circ g} \leftarrow \mathsf{SubkeyGen}(\mathsf{sk}_f, g)$, and assigns this key a unique identity id'. The new value id' is output, and the resulting key $\mathsf{sk}_{f \circ g}$ is kept secret.

  $\mathsf{RevealKey}(id)$: If no key exists with identity id then output $\perp$ and terminate; otherwise output the corresponding key sk_f.

In the challenge phase, the adversary's evaluation queries are answered either (1) consistently pseudorandom, or (2) pseudorandom for all inputs y for which the adversary was given a key sk_f in a $\mathsf{RevealKey}$ query with $y \in \mathit{Range}(f)$, and random for all other inputs.

5.2 Construction Based on OWF

We now construct a functional pseudorandom function family $F_s : \{0,1\}^n \to \{0,1\}^n$ supporting the class of prefix-fixing functions, based on the Goldreich-Goldwasser-Micali (GGM) tree-based PRF construction [GGM86]. More precisely, our construction supports the function class

$$\mathcal{F}_{\mathsf{pre}} = \Big\{ f_z(x) : \{0,1\}^n \to \{0,1\}^n \;\Big|\; z \in \{0,1\}^m \text{ for } m \leq n \Big\},$$

$$\text{where } f_z(x) := \begin{cases} x & \text{if } (x_1 = z_1) \wedge \cdots \wedge (x_m = z_m) \\ \perp & \text{otherwise} \end{cases}.$$

Recall that the GGM construction makes use of a length-doubling pseudorandom generator $G : \{0,1\}^k \to \{0,1\}^{2k}$ (which can be constructed from any one-way function). Denoting the two halves of the output of G as $G(y) = G_0(y)G_1(y)$, the PRF with seed s is defined as $F_s(y) = G_{y_k}(\cdots G_{y_2}(G_{y_1}(s)))$.

We show that we can obtain a functional PRF for $\mathcal{F}_{\mathsf{pre}}$ by adding the following two algorithms on top of the GGM PRF construction. Intuitively, in these algorithms the functional secret key sk_{f_z} corresponding to a queried function $f_z \in \mathcal{F}_{\mathsf{pre}}$ will be the partial evaluation of the GGM prefix corresponding to prefix z: i.e., the label of the node corresponding to node z in the GGM evaluation tree. Given this partial evaluation, a party will be able to compute the completion for any input x which has z as a prefix. However, as we will argue, the evaluation on all other inputs will remain pseudorandom.

$\mathsf{KeyGen}(s, f_z)$: output $G_{z_m}(\cdots G_{z_2}(G_{z_1}(s)))$, where $m = |z|$

$$\mathsf{Eval}(\mathsf{sk}_{f_z}, y) \text{ : output } \begin{cases} G_{y_n}(\cdots G_{y_{m+2}}(G_{y_{m+1}}(\mathsf{sk}_{f_z}))) & \text{if } y_1 = z_1 \wedge \cdots \wedge y_m = z_m \\ \perp & \text{otherwise} \end{cases}$$

We first prove that this construction yields an F-PRF with selective security (i.e., when the adversary's key queries are specified a priori). We then present a sequence of corollaries for achieving full security, PRFs with selective access, and hierarchical F-PRFs. We also focus on the specific application of *punctured* PRFs [SW13].

Theorem 5.5. *Based on the existence of one-way functions, the GGM pseudorandom function family together with algorithms* KeyGen *and* Eval *defined as above, is a selectively secure functional PRF for the class of functions $\mathcal{F}_{\mathsf{pre}}$, as per Definition 5.2.*

Proof. We will reduce the pseudorandom property of our functional PRF scheme to the security of the underlying PRG. Recall that (as per Definition 5.2), the functional PRF requires indistinguishability of experiments Sel-PRand and Sel-Rand, in which the adversary makes key queries (which are answered honestly), and then makes evaluation queries, which are either answered consistently (PRand) or randomly (Rand). At a high level, we will show that both Experiment Sel-Rand and Experiment Sel-PRand are indistinguishable from a third experiment where, in the query phase, the adversary's queries are answered randomly (except when one query is a prefix of another, in which case we need to ensure consistency), and in the challenge phase the adversary's queries are answered randomly. Both claims will be proved using a hybrid argument similar to the proof of the original GGM construction.

Let $f_1, \ldots f_l \in \mathcal{F}_{\mathsf{pre}}$ be the functions queried by the adversary. Let $P_1, \ldots P_l$ be the corresponding prefixes. We consider the following experiments:

Exp 1. Experiment Sel-PRand. In the key query phase, the key for each function f_i corresponding to prefix P_i is obtained (honestly) by following the corresponding path in the GGM tree. In the challenge phase, the adversary's evaluation queries are answered (honestly) with the corresponding pseudorandom values. We denote the probability that an adversary Adv outputs 1 in this experiment by $output^{Adv}_{Exp1}$.

Exp 2. Keys for the queried functions $f_1, \ldots, f_l \in \mathcal{F}_{\mathsf{pre}}$ corresponding to prefixes P_i are computed randomly, up to consistency among queried sub-prefixes. This takes place as follows (recall that all queries are made up front):

- for each f_i, if no prefix of P_i is also queried by the adversary in his keygen queries, then sk_{f_i} is assigned a random value.
- otherwise, let P_j be the shortest such prefix that is also queried (so that sk_{f_j} has already been defined by the previous case). Then sk_{f_i} is computed by honestly applying to sk_{f_j} the sequence of PRG's determined by the bits of P_i following P_j.

In the challenge phase, the adversary's evaluation queries are answered with random values. If a query is repeated, we answer consistently. We denote the probability that an adversary Adv outputs 1 in this experiment by $output^{Adv}_{Exp2}$.

Exp 3 Experiment Sel-Rand. In the key query phase, the key for each function f_i corresponding to prefix P_i is obtained (honestly) by following the corresponding path in GGM tree, and. In the challenge phase, the adversary's evaluation queries (to values not computable by himself already) are answered with random values. If a query is repeated, we answer consistently. We denote the probability that an adversary Adv outputs 1 in this experiment by $output^{Adv}_{Exp3}$.

Note that that experiment described in Exp 1 is Experiment Sel-PRand in the Functional PRF definition, and the experiment described in Exp 3 is Experiment Sel-Rand.

Lemma 5.6. *For any PPT adversary* Adv

$$|output^{Adv}_{Exp1} - output^{Adv}_{Exp2}| = \mathsf{negl}(n).$$

Proof. Suppose there exists an adversary Adv, such that $|output^{Adv}_{Exp1} - output^{Adv}_{Exp2}| = \epsilon(n)$ for some non-negligible $\epsilon(n)$. Wlog, assume that $output^{Adv}_{Exp2} - output^{Adv}_{Exp1} = \epsilon(n) > 0$. We claim that we can use Adv to construct an adversary A_{PRG} that breaks the security of the underlying pseudorandom generator. Recall in the PRG challenge, A_{PRG} receives a polynomial-sized set of values, which are either random or random outputs of the PRG.

We use a hybrid argument, and define Exp^i for $i \in [n]$. The value i corresponds to the level of the tree where A_{PRG} will place his challenge values when interacting with Adv.

In Exp^i, in the key query phase, the key for each function f_j corresponding to prefix P_j of length $|P_j| = m$ is computed as follows:

- if no other queried prefix is a prefix of P_j and $m \leq i$, return a random string of size n.
- if no other queried prefix is a prefix of P_j and $m > i$, set the label of P_j's ancestor on the i^{th} level to a randomly sampled n-bit string, and then apply the pseudorandom generators to it as in the GGM construction according to the remaining bits of P_j until the m^{th} level, and return the resulting string of size n.
- if some other queried prefix is a prefix of P_j, let sk_{f_h} be the key corresponding to the shortest such queried prefix P_h. To obtain the key for P_j, apply the pseudorandom generators to sk_{f_h} as in the GGM construction according to the remaining bits of P_j, up to the m^{th} level of the tree.

In the challenge phase, the answers to the adversary's evaluation queries x are computed as follows:

- let $x^{(i)}$ denote the i-bit prefix of the queried input x. If the node corresponding to $x^{(i)}$ in the tree has not yet been labeled, then a random value is chosen and set as this label. The response to the adversary's query is then computed by applying the PRGs to the label, as determined by the $(i+1)$ to n bits of the queried input x.

Since $output^{Adv}_{Exp2} - output^{Adv}_{Exp1} = \epsilon(n)$, there must exist an i such that:

$$\Pr[\mathsf{Adv} \to 1 \text{ in } Exp^i] - \Pr[\mathsf{Adv} \to 1 \text{ in } Exp^{i+1}] \geq \frac{\epsilon(n)}{n}.$$

Our constructed PRG adversary $\mathsf{A_{PRG}}$ plays the role of the challenger in the game with Adv, chooses a random $i \in [n]$ and places his PRG challenges there. That is, in the key query phase, $\mathsf{A_{PRG}}$ computes the keys for functions f_i corresponding to prefix P_j, of length $|P_j| = m$ as follows:

- if no other queried prefix is a prefix of P_j and $m < i$, return a a random string of size n.
- if no other queried prefix is a prefix of P_j and $m = i$, return one of $\mathsf{A_{PRG}}$'s challenge values.
- if no other queried prefix is a prefix of P_j and $m > i$, set a challenge string as the ancestor of P_j on the i^{th} level, and then apply the pseudorandom generators to it as in the GGM construction until the m^{th} level and return the resulting string of size n.
- if some other queried prefix is a prefix of P_j, let sk_{f_h} be the key corresponding to the shortest such queried prefix, P_h. To obtain the key for P_j, apply the pseudorandom generators to sk_{f_h} as in the GGM construction, up to the m^{th} level of the tree.

In the challenge phase, the answers to the adversary's evaluation queries x are computed as follows:

- let $x^{(i)}$ denote the i-bit prefix of the queried input x. If the node corresponding to $x^{(i)}$ in the tree has not yet been labeled, then one of $\mathsf{A_{PRG}}$'s challenge values is chosen and set as the label. The response to the adversary's query is then computed by applying the PRGs to the label, as determined by the $(i+1)$ to n bits of the queried input x.

Comparing the experiment above to Exp^i and Exp^{i+1}, we can see that, if the inputs to $\mathsf{A_{PRG}}$ are random, $\mathsf{A_{PRG}}$ behaves as the challenger in Exp^i, and if they are the output of a PRG, he behaves as the challenger in Exp^{i+1}.

At the end A_{PRG} outputs the same answer as Adv in its own security game.

$$\Pr[A_{PRG} \text{ guesses correctly }]$$

$$= \frac{1}{2}\Pr[A_{PRG} \to 1 | \text{challenge values random}] + \frac{1}{2}\Pr[A_{PRG} \to 0 | \text{challenge values are output of a PRG}]$$

$$= \frac{1}{2}\Pr[\mathsf{Adv} \text{ outputs 1 in } Exp^i\ ] + \frac{1}{2}\Pr[\mathsf{Adv} \text{ outputs 0 in } Exp^{i+1}]$$

$$= \frac{1}{2}\Pr[\mathsf{Adv} \text{ outputs 1 in } Exp^i\ ] + \frac{1}{2}(1 - \Pr[\mathsf{Adv} \text{ outputs 1 in } Exp^{i+1}])$$

$$= \frac{1}{2} + \frac{1}{2}(\Pr[\mathsf{Adv} \text{ outputs 1 in } Exp^i\ ] - \Pr[\mathsf{Adv} \text{ outputs 1 in } Exp^{i+1}])$$

$$\geq \frac{1}{2} + \frac{\epsilon(n)}{2n}$$

If $\epsilon(n)$ is non-negligible, A_{PRG} can distinguish between random values and outputs of a pseudorandom generator with non-negligible advantage, which would break the security of the underlaying pseudorandom generator. This completes the proof of the lemma.

□

Lemma 5.7. *For any PPT adversary* Adv

$$|output^{Adv}_{Exp2} - output^{Adv}_{Exp3}| = \mathsf{negl}(n).$$

Proof. We use a similar hybrid argument: In Exp^i, in the key query phase, the key for the functions corresponding to prefix P_j, of length $|P_j| = m$ is computed as before:

- if no other queried prefix is a prefix of P_j and $m \leq i$, return a random string of size n.
- if no other queried prefix is a prefix of P_j and $m > i$, set a random string as the parent of P_j on the i^{th} level, and then apply the pseudorandom generators to it as in the GGM construction until the m^{th} level and return the resulting string of size n.
- if some other queried prefix is a prefix of P_j, let sk_{f_h} be the key corresponding to the shortest queried prefix of P_j, P_h. To obtain the key for P_j, apply the pseudorandom generators to sk_{f_h} as in the GGM construction, up to the m^{th} level of the tree.

In the challenge phase, the adversary's queries are answered with random values, unless he has already received a key that allows him to compute the PRF on his queried value, in which case the query is answered consistently.
The first hybrid, Exp^0, is Exp 3, and the last hybrid, Exp^n is Exp 2. □

From the previous lemmas, we can conclude that, for any PPT adversary Adv

$$|output^{Adv}_{Exp1} - output^{Adv}_{Exp3}| = \mathsf{negl}(n).$$

This is equivalent to saying that no PPT adversary can distinguish between Experiment Sel-PRand and Experiment Sel-Rand in the Functional PRF definition. That is, the construction is a secure F-PRF.
□

Remark 5.8. We remark that one can directly obtain a *fully* secure F-PRF for $\mathcal{F}_{\mathsf{pre}}$ (as in Definition 5.1) from our selectively secure construction, with a loss of $\frac{1}{2^n}$ in security for each functional secret key sk_{f_z} queried by the adversary. This is achieved simply by guessing the adversary's query $f_z \in \mathcal{F}_{\mathsf{pre}}$. For appropriate choices of input size n and security parameter k, this can still provide meaningful security.

As an immediate corollary of Theorem 5.5, we obtain a (selectively secure) *PRF with selective access* for the class of equivalent prefix-matching predicates $\mathcal{P}_{\mathsf{pre}} = \{P_z : \{0,1\}^n \to \{0,1\} | z \in \{0,1\}^m \text{ for } m \leq n\}$, where $P_z(x) := 1$ if $(x_1 = z_1) \wedge \cdots \wedge (x_m = z_m)$ and 0 otherwise.

Corollary 5.9. *Based on the existence of one-way functions, the GGM pseudorandom function family together with algorithms* KeyGen *and* Eval *defined as above, is a selectively secure functional PRF for the class of predicates* $\mathcal{P}_{\mathsf{pre}}$.

Our F-PRF construction has the additional benefit of being *hierarchical*. That is, given a secret key sk_{f_z} for a prefix $z \in \{0,1\}^m$, a party can generate subordinate secret keys $\mathsf{sk}_{f_{z'}}$ for any $z' \in \{0,1\}^{m'}$, $m' > m$ agreeing with z on the first m bits. This secondary key generation process is accomplished simply by applying the PRGs to sk_{f_z}, traversing the GGM tree according to the additional bits of z'. We thus achieve the following corollary.

Corollary 5.10. *Based on the existence of one-way functions, the GGM pseudorandom function family together with algorithms* KeyGen *and* Eval *defined as above, is a (selectively secure)* hierarchical *functional PRF for the class of predicates* $\mathcal{P}_{\mathsf{pre}}$.

The pseudorandomness property can be proved using the same techniques as in the proof of Theorem 5.5.

5.2.1 Punctured Pseudorandom Functions

Punctured PRFs, formalized by [SW13], are a special case of functional PRFs where one can generate keys for the function family $\mathcal{F} = \{f_x(y) = y \text{ if } y \neq x, \text{ and } \perp \text{ otherwise}\}$. Such PRFs have recently been shown to have important applications, including use as a primary technique in proving security of various cryptographic primitives based on the existence of indistinguishability obfuscation (see, e.g., [SW13, HSW13]).

The existence of a functional PRF for the prefix-fixing function family gives a construction of punctured PRFs. Namely, a punctured key sk_x allowing one to compute the PRF on all inputs except $x = x_1 \ldots x_n$ consists of n functional keys for the prefix-fixing function family for prefixes: $(\bar{x_1})$, $(x_1\bar{x_2})$, $(x_1x_2\bar{x_3}), \ldots, (x_1x_2 \ldots x_{n-1}\bar{x_n})$.

Our GGM-based construction in the previous section thus directly yields a selectively secure punctured PRF based on OWFs.

Corollary 5.11 (Selectively-Secure Punctured PRFs)**.** *Assuming the existence of OWF, there exists a selectively secure punctured PRF for any desired poly-size input length.*

When considering full security, this may seem an inhibiting limitation, as naïve complexity leveraging over each of the n released keys would incur a tremendous loss in security. However, for a punctured PRF, these n keys are not independently chosen: rather, there is a one-to-one correspondence between the input x that is punctured, and corresponding set of n prefix-fixing keys we give out. This means there are only 2^n possible sets of key queries made by a punctured PRF adversary (as opposed to 2^{n^2} possible choices of n independent prefix queries), and thus, in the full-to-selective security reduction, we lose only a factor of 2^{-n} in the security (as the reduction needs only to guess which of these 2^n query sets will be made by the adversary). Given a desired level of security k and input size $n = n(k)$, and assuming an underlying OWF secure against all adversaries that run in time 2^{K^ϵ} when implemented with security parameter K for some constant $0 < \epsilon < 1$, then by setting $K = n^{1/\epsilon}$, we obtain a fully secure puncturable PRF.

Corollary 5.12. *Assuming the existence of 2^{K^ϵ}-hard OWF for security parameter K and some constant $0 < \epsilon$, there exists a (fully) secure punctured PRF for any desired poly-size input length.*

References

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *CRYPTO*, 2013.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC*, pages 111–120, 2013.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168, 2011.

[BF13] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. Cryptology ePrint Archive, Report 2013/413, 2013.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

[BG89] Mihir Bellare and Shafi Goldwasser. New paradigms for digital signatures and message authentication based on non-interative zero knowledge proofs. In *CRYPTO*, pages 194–211, 1989.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BMS13] Michael Backes, Sebastian Meiser, and Dominique Schrder. Delegatable functional signatures. Cryptology ePrint Archive, Report 2013/408, 2013.

[BSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. Cryptology ePrint Archive, Report 2013/352, 2013.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *FOCS*, pages 308–317, 1990.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS*, pages 553–562, 2005.

[GKP+12] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. *IACR Cryptology ePrint Archive*, 2012:733, 2012.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Overcoming the worst-case curse for cryptographic constructions. In *CRYPTO*, 2013.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377, 1982.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO*, pages 171–185, 1986.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[GW12] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. *IACR Cryptology ePrint Archive*, 2012:290, 2012.

[HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2013/509, 2013.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. Cryptology ePrint Archive, Report 2013/379, 2013.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the dh-ddh separation. In *CRYPTO*, pages 597–612, 2002.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, pages 96–109, 2003.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC*, pages 222–242, 2013.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394, 1990.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW13] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Cryptology ePrint Archive, Report 2013/454, 2013.