Functional Signatures and Pseudorandom Functions

Elette Boyle Shafi Goldwasser Ioana Ivan

June 28, 2013

Abstract

In this paper, we introduce *functional digital signatures* and *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. We show applications of functional signatures to construct 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 $f(x) = y$. This implies the ability to delegate keys per function f for computing a pseudorandom function F on points y for which $f(y) = 1$. We define and provide a sample construction of a functional pseudorandom function family for the prefix-fixing function family.

1 Introduction

We introduce new cryptographic primitives with a variety of accompanying constructions: *functional digital signatures (FDS)*, *functional pseudorandom functions (FPRF)*, and *psuedorandom functions with selective access (PRFSA)*.

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], we require 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 $f_1 \dots f_l$.

Historical Note: This work appeared in part as the Master Thesis of Ioana Ivan filed May 22 at MIT. We note that independently (and unknown to the authors) the notion of pseudorandom functions with selective access that we introduce here was considered by Boneh-Waters under the name of *constrained pseudorandom functions* and by Kiayias, Papadopoulos,Triandopoulos and Zacharias under the name *delegatable pseudorandom functions*.

An application of functional signatures is in certifiable computation. In this setting, there is a client and a server who performs computations for the client. The client gives out keys for a set of functions $\{f_i\}$, and wants to test whether the response from the server is indeed the output of one the f_i on an input. For a concrete example, suppose we have a digital camera that produces signatures of the photos taken with the camera, which can be used to prove that a photo has not been altered. In this case, we might want to allow minor modifications, like changing the color scale, but not allow more significant changes such as merging two photos or cropping a picture. We can use functional signatures to solve this problem, by giving out signing keys for the functions that capture the permissible modifications.

A desirable property from 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. For example, in the example with the signed photos, one might not wish to reveal the original message, just that the final photographs was 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, say, 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) there exists a polynomial-time algorithm that given an index s (which can be thought of as a secret key) can evaluate $F_s(x)$ for every x in the function domain, but (2) no probabilistic polynomial-time algorithm *without* s who can request and receive evaluations $F_s(x_i)$ for inputs x_i's of its choice can distinguish $(z, F_s(z))$ from (z, R) with R random for any $z \neq x_i$ not previously seen.

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, the following paradigm can be followed: 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. The latter had been used as a way to construct families of digital signature in [BG89].

In this work, we extend pseudorandom functions to a primitive which we call *functional pseudorandom functions (FPRF)*. 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 as follows. 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 public committed input and (2) use y as her next sequence of coins in the protocol. To prove that the appropriate randomness was used, id can utilize NIZK proofs.[1] 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 FPRF appealing in its simplicity and efficiency.

The notion of functional pseudorandom functions has many variations. One natural variant which 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 FPRF, 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. Another variant is *hierarchical pseudorandom functions*, with an additional property that parties with functional keys sk_f

[1] An interesting open question is how to achieve a *verifiable* FPRF, where there is additional information vk_s that can be used to verify that a given pair$(x, F_s(x))$ is valid. This would eliminate the need for the NIZK proofs.

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.

1.1 Summary of Our Results on Functional Signature

We provide a construction of functional signatures which achieves 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.

Theorem 1 (Informal)**.** *Assuming the existence of succinct non interactive-arguments of knowledge (SNARKs) and NIZKAoK for NP languages, there exists a succinct 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 and function privacy. The size of the signature only depends on the security parameter and the size of $f(m)$.*

Overview of the construction:
Assuming the existence of one-way functions, Rompel constructs a signature scheme that is existentially unforgeable under chosen message attack in [Rom90].

In the setup algorithm for our functional signature scheme, we sample a key pair $(\mathsf{msk}, \mathsf{mvk})$ for the standard signature scheme, and set the master signing key for the functional signature scheme to be msk, and the master verification key to be mvk.

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, and the running time of the verifier is sublinear in the running time of R. SNARK schemes have been constructed under various non-falsifiable assumptions. For example, Bitansky et al. [BCCT12] 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, the size of the instance, and the logarithm of the time it takes to verify a valid witness for the instance assuming the existence of extractable collision resistance hash functions and NIZKAoK. (They also show that any SNARK + NIZKAoK directly yield zero-knowledge (ZK)-SNARK with analogous parameters). More details are given in Section 2.3.

To generate a signing key for a function f, we output a signature of f under mvk as the signing key sk_f. To sign a message $m^* = f(m)$ using sk_f, we generate a zero-knowledge SNARK for the following statement: $\exists(\sigma, f, m)$ such that $m^* = f(m)$ and σ is a valid signature of f under mvk. To verify the signature, we run the verification algorithm for the SNARK argument.

Resorting to non-falsifiable assumptions, albeit strong, seems necessary to obtain succinctness for functional signatures. We see this as follows. Recall that in a succinct non-interactive arguments (SNARG) for a language L, there is a verifier V, and a prover P who is attempting to convince the verifier that an input $x \in L$. Let us require the proofs produced by prover to be sublinear in the size of the input plus the size of the witness. Then, we show

Theorem 2 (Informal)**.** *If there exists a functional signature that supports key for any function f, and has short signatures (i.e of size $poly(k) \cdot o(|f(m)| + |m|)$, then there exists a SNARG scheme with preprocessing for any language $L \in NP$ with proof size $poly(k) \cdot o(|w| + |x|)$, where w is the witness and x is the instance.*

As Gentry and Wichs show in [GW11] that SNARG schemes with proof size $o(|w| + |x|)$ cannot be obtained using black-box reductions to falsifiable assumptions, we can conclude that in order to obtain a functional signature scheme with signature size $o(|f(m)| + |m|)$ we must either rely on non-falsifiable assumptions (as in our SNARK construction) or make use of non blackbox techniques.

We remark that if one was willing to give up on the requirement of succinctness and function privacy for a functional signature scheme, one could obtain constructions requiring only the existence of one-way functions.

Theorem 3 (Informal)**.** *Assuming the existence of one-way functions, 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:
Assuming the existence of one-way functions, Rompel constructs a signature scheme that is existentially unforgeable under chosen message attack in [Rom90].

In the setup algorithm for our functional signature scheme, we sample a key pair $(\mathsf{msk}, \mathsf{mvk})$ for the standard signature scheme, and set the master signing key for the functional signature scheme to be msk, and the master verification key to be mvk.

To generate a signing key for a function f, we sample a new signing and verification key pair $(\mathsf{sk}', \mathsf{vk}')$, and sign the concatenation of f and vk', $f|\mathsf{vk}'$ using msk. The signing key for f consists of this certificate 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 $f|\mathsf{vk}'$ under msk.

Finally, a scheme which satisfies functional privacy but where the size of a signature output by the algorithm $\mathsf{Sign}(\mathsf{sk_f}, m)$ depends on the size of a circuit computing f, can be based on the existence of non-interactive zero-knowledge arguments of knowledge (NIZKAoK) for NP.

Theorem 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 both the unforgeability requirement for functional signatures and function privacy, but not succinctness. The size of the signature is dependent on the size of f and m.*

Overview of the construction:
The setup algorithm for the functional signature scheme is the same as in the SNARK-based scheme above: we sample a key pair $(\mathsf{msk}, \mathsf{mvk})$ for the standard signature scheme, and set the master signing key for the functional signature scheme to be msk, and the master verification key to be mvk.

In the key generation algorithm for a function f, we output the signature of f under mvk as the signing key $\mathsf{sk_f}$. To sign a message $m^* = f(m)$ using $\mathsf{sk_f}$, we generate a NIZKAoK for the following statement: $\exists(\sigma, f, m)$ such that $m^* = f(m)$ and σ is a valid signature of f under mvk. To verify the signature, we run the verification algorithm for the NIZKAoK argument system.

While this signature satisfies both unforgeability and function privacy, that size of a signature is still polynomial in the security parameter, $|m|$, and $|f|$.

Relation to Delegation: As evident by the constructions above, functional signatures are highly related to delegation schemes. 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 so that the the time to verify is more efficient than computing the function. Indeed, we show that given *any* functional signature scheme supporting a class of functions $\mathcal{F}$, with signature size $s(k)$, and verification time $t(k)$ we can obtain a delegation scheme in the preprocessing model for functions in $\mathcal{F}$, with proof size $s(k)$ and verification time $t(k)$.

Theorem 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 $t(k)$.*

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

We present formal definitions and constructions of functional pseudorandom functions (FPRF) and pseudorandom functions with selective access (PRFSA). 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*, based on the Goldreich-Goldwasser-Micali (GGM) tree-based PRF construction [GGM86].

Theorem 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 or any of the functions.*

We remark that one can directly obtain a *fully* secure FPRF for $\mathcal{F}_{\mathsf{pre}}$, in which security holds against the 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 $f_z \in \mathcal{F}_{\mathsf{pre}}$.

Overview of the construction. We show that the original Goldreich-Goldwasser-Micali (GGM86) 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 beginning 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). In Section 5, we prove that in fact security is maintained even when several secret keys sk_{f_z} are queried.

Our construction has the additional beneficial property of *hierarchical key generation*: i.e., a party with a functional key sk_f for a function f may generate valid "subordinate" functional keys sk_g for any function $g = f' \circ f$. That is, we prove the following additional statement.

Corollary 7 (Informal)**.** *Assuming the existence of OWF, there exists a* hierarchical *functional PRF for the class of functions F_{pre}.*

As an immediate corollary of the above, we achieve (hierarchical) functional PRFs with *selective access* for the corresponding class of prefix-matching predicates:

Corollary 8 (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

Constructing functional signatures with short (sublinear in the size of the functions supported) signatures and verification time under falsifiable assumptions remains an open problem. 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 blackbox techniques. No lower bound exists that relates the size of the signature to the description of f.

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. In the past few years there has been significant progress on the problem of functional encryption (e.g., [GVW12, GKP$^+$12,

GKP+13]). In this setting, a center with access to a master secret key can generate a secret key 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:

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

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

Papamantou, 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

This work appeared in part as the Master Thesis of Ioana Ivan. 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* and by Kiayias, Papadopoulos,Triandopoulos and Zacharias under the name *delegatable pseudorandom functions*.

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 constructions for several function families

2 Preliminaries

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

2.1 Signature Schemes

Definition 9. A signature scheme for a message space $\mathcal{M}$ is a tuple $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$:

- $\mathsf{Gen}(1^k) \rightarrow (\mathsf{sk}, \mathsf{vk})$: 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 $(\mathsf{sk}, \mathsf{vk})$.
- $\mathsf{Sign}(\mathsf{sk}, m) \rightarrow \sigma$: 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.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \rightarrow \{0, 1\}$: 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

$$\forall \sigma \in \mathsf{Sign}(\mathsf{sk}, m), \mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1$$

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 a signature for message of his choice, and the challenger responds with a signature. This is repeated a polynomial number of times. Each query can be chosen adaptively, based on vk, and the signatures received for the previous queries.
- The adversary outputs a signature σ^*, and wins if there exists a message m^* such that $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$, and the adversary has not previously received a signature of m^* from the challenger.

Lemma 10 ([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 11. [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 $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$,

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

- **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 $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$, is at most $\mu(k)$.
- **Adaptive Zero-Knowledge**: For all PPT adversaries $\mathcal{A}$,

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

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

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

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

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

  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 12. 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 PPT adversary $\mathcal{A}$,

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

and for every PPT adversary $\mathcal{A}$,

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

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.

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 $c = \mathsf{Enc}_{\mathsf{pk}}(w, r)$ and compute a NIZK proof π, using the underlying NIZK argument system Π, that $(\mathsf{pk}, x, c) \in L'$, where

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

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.

Lemma 13 ([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 14. $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ 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**: 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**: The length of a proof is given by $|\pi| = poly(k) \cdot o(|x| + |w|)$. In this work we will refer to a SNARG as having proof length $poly(k) \cdot \mathsf{poly}(|x| + \log R)$, where R denotes the runtime of the relation associated with language L.

Definition 15. 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 f there exists a PPT algorithm $\mathsf{E} = (\mathsf{E}_1, \mathsf{E}_2)$ such that for every PPT adversary $\mathcal{A}$,

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

and for every PPT adversary $\mathcal{A}$,

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

$$= \mathsf{negl}(k).$$

Definition 16. 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 $\mathcal{A}$,

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

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).

For example, in [BCCT12] Bitansky et al. give a construction of SNARKs assuming the existence of extractable collision-resistant hash functions.

Lemma 17 ([BCCT12])**.** *If there exist ECRHs and then there exist SNARKs for all languages in NP.*

Lemma 18 ([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 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.

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.

Definition 19. A *delegation scheme* for a function F consists of a tuple of algorithms (KeyGen, Encode, $\mathsf{Compute}$, Verify)

- $\mathsf{KeyGen}(1^k, F) \to (\mathsf{enc}, \mathsf{evk}, \mathsf{vk})$: 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.
- $\mathsf{Encode}(\mathsf{enc}, x) \to \sigma_x$: 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.
- $\mathsf{Compute}(\mathsf{evk}, \sigma_x) \to (y, \pi_y)$: 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.
- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi_y) \to \{0, 1\}$: 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 $(y, \pi_y) \leftarrow \mathsf{Compute}(\mathsf{evk}, \sigma_x)$, $\sigma_x \leftarrow \mathsf{Encode}(\mathsf{enc}, x)$, $(\mathsf{enc}, \mathsf{evk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^k, F)$,

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

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

- The challenger runs $\mathsf{KeyGen}(1^k, F) \to (\mathsf{enc}, \mathsf{evk}, \mathsf{vk})$, 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 $\mathsf{Verify}(\mathsf{vk}, x, y, \pi_y) = 1$.

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 20. A *pseudorandom generator* (PRG) is a length expanding function $\mathsf{prg} : \{0,1\}^k \to \{0,1\}^n$ (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 21. [GGM86] A family of functions $\mathcal{F} = \{F_s\}_{s \in S}$, indexed by a set S, and where $F_s : D \to R$ for all s, is a *pseudorandom function (PRF) family* if for a randomly chosen s, and all PPT $\mathcal{A}$, the distinguishing advantage $\Pr_{s \leftarrow S}[\mathcal{A}^{f_s(\cdot)} = 1] - \Pr_{f \leftarrow (D \to R)}[\mathcal{A}^{\rho(\cdot)} = 1]$ is negligible, where $(D \to R)$ 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 satisfies.

Definition 22. A *functional signature scheme* for a message space $\mathcal{M}$ 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 KeyGen algorithm takes as input the master signing key and a function f (represented as a circuit), and outputs a signing key for f.
- $\mathsf{FS.Sign}(\mathsf{sk_f}, m) \to (f(m), \sigma)$: the signing algorithm takes as input the signing key for a function f and an input m, 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:
$\forall m, f, (\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{FS.Setup}(1^k), \mathsf{sk_f} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f), (m^*, \sigma) \leftarrow \mathsf{FS.Sign}(\mathsf{sk_f}, m)$,

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

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 $\mathsf{O_{key}}(f) = \mathsf{FS.KeyGen}(\mathsf{msk}, f)$, and a signing oracle $\mathsf{O_{sign}}(f, m) = \mathsf{FS.Sign}(\mathsf{sk_f}, m)$, where $\mathsf{sk_f} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f)$.
- 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:
The advantage of any PPT adversary in the following game is negligible:

- The adversary is given mvk and access to the $\mathsf{O_{key}}$ and $\mathsf{O_{sign}}$ oracles, as in the unforgeability game.
- After the query phase, the adversary chooses m_0, m_1, f_0, f_1 such that $f_0(m_0) = f_1(m_1)$ and sends them to the challenger.
- The challenger chooses $b \in \{0, 1\}$ and gives the adversary $\mathsf{FS.Sign}(\mathsf{sk_{f_b}}, m_b)$, where $\mathsf{sk_{f_b}} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f_b)$.
- The adversary wins the game if he guesses the bit b correctly.

Succinctness:
The size of a signature $\sigma \leftarrow \mathsf{FS.Sign}(\mathsf{sk_f}, m)$ is bounded by a polynomial in the security parameter k, and the size of the output $|f(m)|$. In particular, it is independent of $|m|$, the size of the input to the function, and $|f|$, the size 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 proofs of knowledge (SNARKs).

Theorem 23. *Assuming the existence of SNARKs for NP, there exists a function-private functional signature scheme for the class of all polynoial-size circuits, such that both the size of a signature corresponding to function f and input m, and the running time of the verification algorithm, are $poly(k, |f(m)|)$ (where k is the security parameter).*

We obtain two other constructions, which satisfy weaker properties, but are based on lighter assumptions. If we do not require the signatures to be succinct, we give a construction based on non-interactive zero-knowledge arguments of knowledge (NIZKAoKs). Finally, in the third section, we obtain a construction based on any one-way function that achieves basic correctness and unforgeability but yields non-succinct signatures and does not provide function privacy.

We present these three constructions in the following three subsections.

3.2.1 SNARK-Based Construction

In this section, we discuss our main construction: a functional signature scheme that is secure under less standard assumptions, but achieves the desired unforgeability, function privacy, and succinctness requirements. In particular, the size of a signature associated with function f and input m, and the running time of the verification algorithm, are now polynomial in the security parameter and $|f(m)|$, instead of $|m| + |f|$. Our construction is based on SNARKs.

Theorem 24 ([BCCT12])**.** *If there exist extractable collision resistant hash function, there exist SNARKs. If there exist SNARKs and (standard) NIZKAoKs, there exist zero-knowledge SNARKs.*

Let $\mathsf{Sig} = (\mathsf{Sig.Setup}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be a signature scheme that is existentially unforgeable under chosen message attack. 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:
$x = (m^*, \mathsf{mvk}) \in L$ if $\exists (f, m, \sigma)$ such that:

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

Given the signature scheme Sig and the zero-knowledge SNARK Π, we construct a functional signature scheme $(\mathsf{FS1.Setup}, \mathsf{FS1.Keygen}, \mathsf{FS1.Sign}, \mathsf{FS1.Verify})$ as follows:

- $\mathsf{FS1.Setup}(1^k)$:
  - choose a new signing, verification key pair for the regular signature scheme $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.Setup}(1^k)$.
  - choose a new crs for the zero-knowledge SNARK, $\mathsf{crs} \leftarrow \Pi.\mathsf{Gen}(1^k)$.
  - set the master secret key $\mathsf{msk} = \mathsf{sk}$, and the master verification key $\mathsf{mvk} = (\mathsf{vk}, \mathsf{crs})$.
- $\mathsf{FS1.KeyGen}(\mathsf{msk}, f)$:
  - create a certificate consisting of f, and a signature of f under the master verification key: $c = (f, \mathsf{Sig.Sign}(\mathsf{msk}, f))$.
  - output $\mathsf{sk_f} = c$
- $\mathsf{FS1.Sign}(sk_f, m)$:
  - let $\pi = \Pi.\mathsf{Prove}((f(m), \mathsf{mvk}), (f, m, \mathsf{sk_f}), \mathsf{crs})$ be a zero-knowledge SNARK that $(f(m), \mathsf{mvk}) \in L$, where L is defined as above. Informally, π is a proof that the signer knows a pair (f, m) such that $f(m) = m^*$, and also knows a signature of f under the master verification key.
  - output $(m^* = f(m), \sigma = \pi)$
- $\mathsf{FS1.Verify}(\mathsf{mvk}, m^*, \sigma)$:
  - output $\Pi.\mathsf{Verify}(crs, m^*, \sigma)$: verify that σ is a valid proof of knowledge of a pair (f, m) such that $f(m) = m^*$, and a signature of f under the master verification key.

Theorem 25. *If the signature scheme (*$\mathsf{Sig.Setup}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify}$*) is existentially unforgeable under chosen message attack, and* $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}), \mathsf{E} = \mathsf{E}_1, \mathsf{E}_2)$ *is a zero-knowledge (ZK) SNARK for the NP language L, then the scheme (*$\mathsf{FS1.Setup}, \mathsf{FS1.KeyGen}, \mathsf{FS1.Sign}, \mathsf{FS1.Verify}$*) as specified above satisfies the unforgeability, function privacy, and succinctness requirements for functional signatures.*

Proof. **Proof of unforgeability**
Suppose there exists an adversary $\mathsf{A_{FS}}$ that produces a forgery in the functional signature scheme with non-negligible probability. We show how to construct an adversary $\mathsf{A_{sig}}$ that uses $\mathsf{A_{FS}}$ to produce a forgery in the underlying signature scheme.

In the security game for the standard (existentially unforgeable under chosen message attack) signature scheme, $\mathsf{A_{sig}}$ is given the verification key vk, and access to a signing oracle $\mathsf{O}_{\mathsf{Reg_{sig}}}$. He is considered to be successful in producing a forgery if he outputs a valid signature for a message that was not queried from $\mathsf{O}_{\mathsf{Reg_{sig}}}$.

We now construct the adversary $\mathsf{A_{sig}}$. $\mathsf{A_{sig}}$ interacts with $\mathsf{A_{FS}}$, playing the role of the challenger in the security game for the functional signature scheme. $\mathsf{A_{sig}}$ generates $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{E}_1(1^k)$, a *simulated* CRS for the ZK-SNARK, together with a trapdoor, and forwards $(\mathsf{vk}, \mathsf{crs})$ as the master verification key in the functional signature scheme to $\mathsf{A_{FS}}$.
$\mathsf{A_{FS}}$ makes two types of queries:

- $\mathsf{O}_{\mathsf{key}}(f)$, which $\mathsf{A_{sig}}$ answers (honestly) by forwarding f to its signing oracle.
- $\mathsf{O}_{\mathsf{sign}}(f, m)$, in which case $\mathsf{A_{sig}}$ forwards to the signing oracle the function f', which is the *constant function* that outputs $f(m)$ on any input, and receives a signature $\sigma \leftarrow \mathsf{O}_{\mathsf{Reg_{sig}}}(f')$. It then outputs $\pi \leftarrow \Pi.\mathsf{Prove}((f(m)\mathsf{mvk}), (f', f(m), \sigma), \mathsf{crs})$ as its signature of $f(m)$.

  Note that this is not the honest response, which would have answered with respect to the queried function f (which may contain additional values in its range) instead of the constant function f' (whose range consists of only $f(m)$).

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

We now prove that if $\mathsf{A_{FS}}$ forges with noticeable probability in the functional signature scheme then this constructed adversary $\mathsf{A_{sig}}$ produces a successful forgery in the underlying signature scheme with noticeable probability. We do so by considering a sequence of hybrid experiments:

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 $\mathsf{O}_{\mathsf{sign}}(f, m)$ are answered honestly, by first generating a certificate for the queried function f (instead of the constant function $f' \equiv f(m)$) and continuing appropriately. Denote the probability of the adversary producing a valid forgery in the functional signature scheme within this experiment by Forge_0.

Hybrid 1. Similar to Hybrid 0, except that the adversary's signature queries $\mathsf{O}_{\mathsf{sign}}(f, m)$ are now answered with respect to the *constant* function $f' \equiv f(m)$, as described above. Namely, for each signing query $\mathsf{O}_{\mathsf{sign}}(f, m)$ made by the adversary, a signature $\sigma_{f'} \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f')$ is generated on f' (not f) in the underlying signature scheme, a ZK-SNARK proof is generated as $\pi \leftarrow \Pi.\mathsf{Prove}((f(m), \mathsf{mvk}), (f, m, \sigma_{f'}), \mathsf{crs})$, and then π is returned as the query response. All key queries $\mathsf{O}_{\mathsf{key}}(f)$ made by the adversary are still answered honestly, by generating a signature $\sigma_f \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f)$ and returning σ_f.

Denote the probability of the adversary producing a valid forgery in the functional signature scheme within this experiment by Forge_1.

Hybrid 2. The same experiment as Hybrid 1, 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_2.

Hybrid 3. The interaction with the adversary is the same as in Hybrid 2. Denote by M the set of all messages signed with msk in the underlying signature scheme during the course of the experiment,

as a result of the adversary'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., $(f^*, m, \sigma^*) \leftarrow \mathsf{E}_2(\mathsf{crs}, \mathsf{trap}, (m^*, \mathsf{vk}), \pi^*)$.

Denote by $\mathsf{Extract}_3$ the probability that σ^* is a valid signature on a function f^* such that $f^* \notin M$. Note that this corresponds to the probability of $\mathsf{A}_{\mathsf{sig}}$ successfully producing a forgery in the underlying signature scheme.

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

Lemma 26. $\mathsf{Forge}_0 \leq \mathsf{Forge}_1 + negl(k)$.

Proof. This witness indistinguishability property is implied by the zero knowledge property of the ZK-SNARK system. Namely, denoting the queried values as (f_i, m_i) and the corresponding constant functions as $f'_i \equiv f_i(m_i)$, the zero knowledge property (Definition 16) implies that both distributions of valid proofs with different witnesses

$$\Big\{(\mathsf{crs}, \pi_1, \ldots, \pi_\ell) : \mathsf{crs} \leftarrow \mathsf{Gen}_1(1^k), \sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f_i), \pi_i \leftarrow \mathsf{Prove}\big((f_i(m_i), \mathsf{mvk}), (f_i, m_i, \sigma_i), \mathsf{crs}\big)\Big\},$$

$$\Big\{(\mathsf{crs}, \pi_1, \ldots, \pi_\ell) : \mathsf{crs} \leftarrow \mathsf{Gen}_1(1^k), \sigma'_i \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f'_i), \pi_i \leftarrow \mathsf{Prove}\big((f_i(m_i), \mathsf{mvk}), (f'_i, f_i(m_i), \sigma'_i), \mathsf{crs}\big)\Big\}$$

are computationally indistinguishable from a third, simulated distribution

$$\Big\{(\mathsf{crs}, \pi_1, \ldots, \pi_\ell) : (\mathsf{crs}, \mathsf{trap}^{\mathsf{sim}}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k), \pi_i \leftarrow \mathcal{S}^{\mathsf{proof}}\big((f_i(m_i), \mathsf{mvk}), \mathsf{trap}^{\mathsf{sim}}, \mathsf{crs}\big)\Big\},$$

and thus indistinguishable from each other. Thus, since the event of successfully forging in the functional signature challenge is publicly testable, the lemma must hold. □

Lemma 27. $\mathsf{Forge}_1 \leq \mathsf{Forge}_2 + negl(k)$.

Proof. Follows directly by the indistinguishability of CRS values generated via the standard algorithm Gen and the extraction-enabling algorithm E_1, as per Definition 15.

More formally, suppose there exists a PPT adversary $\mathcal{A}$ for which $\mathsf{Forge}_2 < \mathsf{Forge}_1 - \epsilon$ for some ϵ. Then the following 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 $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.Setup}(1^k)$ for the underlying signature scheme, and sends $\mathsf{mvk} = (\mathsf{vk}, \mathsf{crs})$ to $\mathcal{A}$. He answers $\mathcal{A}$'s queries as in Hybrid 1, generating signatures and proofs as required (note that $\mathcal{A}$ holds the master secret key $\mathsf{msk} = \mathsf{sk}$ for the functional signature scheme). At the conclusion of $\mathcal{A}$'s queries, he outputs an alleged forgery π^* in the functional signature scheme. The adversary $\mathcal{A}_{\mathsf{crs}}$ tests whether π^* is indeed a forgery (note that this is publicly testable). If so, $\mathcal{A}_{\mathsf{crs}}$ outputs "standard crs"; otherwise, he outputs "extractable crs". His advantage in the CRS distinguishing game is precisely $\mathsf{Forge}_2 - \mathsf{Forge}_1$, as desired. □

Lemma 28. $\mathsf{Forge}_2 \leq \mathsf{Extract}_3 + negl(k)$.

Proof. This holds by the extraction property of the ZK-SNARK system (Definition 15).

Namely, if there exists a PPT adversary $\mathcal{A}$ for which $\mathsf{Forge}_2 > \mathsf{Extract}_3 + \epsilon$ 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 $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.Sign}(1^k)$ for the underlying signature scheme, sends $\mathsf{mvk} = (\mathsf{vk}, \mathsf{crs})$ to the adversary $\mathcal{A}$, and answers all of $\mathcal{A}$'s key and signing oracle queries as in Hybrid 2. In particular, for each key query $\mathsf{O}_{\mathsf{key}}$, the adversary $\mathcal{A}_{\mathsf{Ext}}$ responds with a signature $\sigma_f \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, f)$; for each signature query $\mathsf{O}_{\mathsf{sign}}(f, m)$, the adversary $\mathcal{A}_{\mathsf{Ext}}$ samples a signature $\sigma_{f'} \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, f')$ on the *constant function* $f' \equiv f(m)$ (as in Hybrid 2) and honestly generates a proof $\pi \leftarrow \Pi.\mathsf{Prove}((f(m), \mathsf{vk}), (f, m, \sigma_f), \mathsf{crs})$, which he returns to $\mathcal{A}$. At the conclusion of interaction, $\mathcal{A}$ outputs an alleged forgery (m^*, π^*). The adversary $\mathcal{A}_{\mathsf{Ext}}$ outputs the proof π^* as his response in the SNARK extraction challenge.

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 $(m^*, \mathsf{vk}) \in L$. We argue that if extraction succeeds on π^* (i.e. if

$(f^*, m, \sigma^*) \leftarrow \mathsf{E}_2(\mathsf{crs}, \mathsf{trap}, (m^*, \mathsf{vk}), \pi^*)$ 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 $g \notin M$, so that we are in the event corresponding to $\mathsf{Extract}_3$. That is, we show $\mathsf{Forge}_2 - \mathsf{Extract}_3$ is bounded above by the probability that extraction *fails*.

Since π^* is a valid forgery in the functional signature scheme, it must be that $m^* \notin Range(g)$ for all key queries $\mathsf{O}_{\mathsf{key}}(g)$ made by $\mathcal{A}$, and that $m^* \neq g(x)$ for all signing queries $\mathsf{O}_{\mathsf{sign}}(g, x)$ 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 29. $\mathsf{Extract}_3 < negl(k)$.

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

Proof of function privacy
For an adversary to win the function privacy game, it must be able to distinguish between $\pi_1 \leftarrow \Pi.\mathsf{Prove}((m, \mathsf{mvk}), (f_1, m_1, \sigma_1), \mathsf{crs})$ and $\pi_2 \leftarrow \Pi.\mathsf{Prove}((m, \mathsf{mvk}), (f_2, m_2, \sigma_2), \mathsf{crs})$, where $m = f_1(m_1) = f_2(m_2)$ and $\sigma_1$is a valid signature of f_1, and $\sigma_2$is a valid signature of f_2. Such an adversary would then directly break the zero-knowledge property of the proof system, since it can distinguish between proofs generated using different witnesses. Hence, function privacy must hold.

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 $\mathsf{FS1.Sign}(\mathsf{sk}_f, m)$ is exactly the proof length of a SNARK for the language L. Since a statement in L is $(f(m), \mathsf{vk})$ and the relation testing runtime is $|f| + poly(k)$, by Definition 14 the corresponding proof length is bounded by $poly(k) \cdot poly(|f(m)| + |\mathsf{vk}| + \log(|f| + poly(k)))$. We may assume $|f| = poly(k)$, which implies the signature size is $poly(k) \cdot poly(|f(m)|)$, as desired.

□

3.2.2 NIZK-based construction

In order to obtain a functional signature scheme that only satisfies unforgeability and function privacy under more general assumptions, we modify the previous construction to use standard non-interactive zero-knowledge proofs of knowledge (NIZKAoK). We remark that our construction hides the function f, but it reveals the size of a circuit computing f.

Let ($\mathsf{FS2.Setup}$, $\mathsf{FS2.Keygen}$, $\mathsf{FS2.Sign}$, $\mathsf{FS2.Verify}$) be a functional signature scheme which is identical to our previous construction $\mathsf{FS1}$, except that we use a NIZKAoK Π', instead of the zero-knowledge SNARK system Π.

Theorem 30. *If (*$\mathsf{Sig.Setup}$, $\mathsf{Sig.Sign}$, $\mathsf{Sig.\ Verify}$*) is an existentially unforgeable signature scheme, and* Π' *is a NIZKAoK, our new functional signature construction (*$\mathsf{FS2.Setup}$, $\mathsf{FS2.Keygen}$, $\mathsf{FS2.Sign}$, $\mathsf{FS2.Verify}$*) 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 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.

3.2.3 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 functional signature scheme satisfies the unforgeability property given in Definition 22, but not function privacy or succinctness. Since we can build standard signature schemes 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 idea in this construction is that, as part of the signing key for a function f, the signer receives from the central authority a signature of f together with a new verification key (under the master verification key). We can think of this signature as a certificate proving that the signer has received permission to sign messages that are in the range of f.

We describe the construction below:

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

- $\mathsf{FS3.Setup}(1^k)$:
  - 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.
- $\mathsf{FS3.KeyGen}(\mathsf{msk}, f)$:
  - choose a new signing and verification key pair for the original signature scheme: $(\mathsf{sk}'_\mathsf{f}, \mathsf{vk}'_\mathsf{f}) \leftarrow \mathsf{Sig.Setup}(1^k)$.
  - using msk, compute $\sigma'' \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f|\mathsf{vk}'_\mathsf{f})$, a signature of f concatenated with the new signing key vk'_f.
  - create the certificate $c = (f, \mathsf{vk}'_\mathsf{f}, \sigma'')$.
  - output $\mathsf{sk}_\mathsf{f} = (\mathsf{sk}'_\mathsf{f}, c)$.
- $\mathsf{FS3.Sign}(\mathsf{sk}_\mathsf{f}, m)$:
  - parse sk_f as $(\mathsf{sk}'_\mathsf{f}, c)$, where sk'_f is a signing key for the existentially unforgeable signature scheme, and c is a certificate as described in the KeyGen algorithm.
  - sign m using sk'_f: $\mathsf{Sig.Sign}(\mathsf{sk}'_\mathsf{f}, m) \to \sigma'$.
  - let $\sigma = (m, c, \sigma')$
  - output $(f(m), \sigma)$
- $\mathsf{FS3.Verify}(\mathsf{mvk}, m^*, \sigma)$:
  - parse $\sigma = (m, c = (f, \mathsf{vk}'_\mathsf{f}, \sigma''), \sigma')$ and check that:
    1. $m^* = f(m)$.
    2. $\mathsf{Sig.Verify}(\mathsf{vk}'_\mathsf{f}, m, \sigma') = 1$: σ' is a valid signature of m under the verification key vk'_f.
    3. $\mathsf{Sig.Verify}(\mathsf{mvk}, \mathsf{vk}'_\mathsf{f}|f, \sigma'') = 1$: σ'' is a valid signature of $f|\mathsf{vk}'_\mathsf{f}$ under the verification key mvk.

Theorem 31. *If the signature scheme (*$\mathsf{Sig.Setup}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify}$*) is existentially unforgeable under chosen message attack, the functional signature scheme (*$\mathsf{FS3.Setup}$, $\mathsf{FS3.KeyGen}$, $\mathsf{FS3.Sign}$, $\mathsf{FS3.Verify}$*) as specified above satisfies the unforgeability requirement for functional signatures.*

Proof. Suppose there exists an adversary $\mathsf{A_{FS}}$ that makes at most Q(k) queries to the $\mathsf{O_{key}}$ and $\mathsf{O_{sign}}$ oracles, and wins the unforgeability game for functional signatures with non-negligible probability, $\frac{1}{P(k)}$, where P and Q are polynomials. We will use him to construct an adversary $\mathsf{A_{sig}}$ that breaks the underlaying signature scheme, which is assumed to be secure against chosen message attack.

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

- σ' is a valid signature of m under the verification key vk'_f.
- σ'' is a valid signature of $f|\mathsf{vk}'_\mathsf{f}$ under mvk.

- $f(m) = m^*$.
- $\mathsf{A_{FS}}$ has not sent the query $\mathsf{O_{key}}(\tilde{f})$ to the signing key generation oracle for any $\tilde{f}$ that has m^* in its range.
- $\mathsf{A_{FS}}$ hasn't sent the query $\mathsf{O_{sign}}(\tilde{f}, \tilde{m})$ to the signing oracle for any $\tilde{f}, \tilde{m}$ such that $\tilde{f}(\tilde{m}) = m^*$

There are two ways $\mathsf{A_{FS}}$ can produce a forgery:

- **Type I forgery**: $\mathsf{A_{FS}}$ produces a signature σ'' of $(f|\mathsf{vk}'_\mathsf{f})$ under mvk, for a function f not queried from the $\mathsf{O_{key}}$ oracle.
- **Type II forgery**: $\mathsf{A_{FS}}$ obtains $\mathsf{Sig.Sign}(\mathsf{msk}, f|\mathsf{vk}'_\mathsf{f})$, and $\mathsf{Sig.Sign}(\mathsf{sk}'_\mathsf{f}, m)$ as part of a query $\mathsf{O_{sign}}(f, m)$ to the signing oracle, and then forges $\mathsf{Sig.Sign}(\mathsf{sk}'_\mathsf{f}, m')$, for a different message m'.

In the security game for the standard (existentially unforgeable under chosen message attack) signature scheme, $\mathsf{A_{sig}}$ is given the verification key vk, and access to a signing oracle $\mathsf{O_{Reg_{sig}}}$. He is considered to be successful in producing a forgery if he outputs a valid signature for a message that was not queried from $\mathsf{O_{Reg_{sig}}}$.

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

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

- first $\mathsf{A_{sig}}$ gives vk to $\mathsf{A_{FS}}$ as the master verification key.
- to answer a key generation query for a function f, $\mathsf{A_{sig}}$ generates a new key pair for the regular signature scheme, $(\mathsf{sk}'_\mathsf{f}, \mathsf{vk}'_\mathsf{f}) \leftarrow \mathsf{Sig.Setup}(1^k)$, forwards $(f|\mathsf{vk}'_\mathsf{f})$ to its signing oracle, obtains $\sigma'' \leftarrow \mathsf{O_{Reg_{sig}}}(f|\mathsf{vk}'_\mathsf{f})$ and returns $\mathsf{sk_f} = (\mathsf{sk}'_\mathsf{f}, \sigma'')$ to $\mathsf{A_{FS}}$.
- to answer a signing query for (f, m), $\mathsf{A_{sig}}$ chooses a new signing, verification key pair $(\mathsf{sk}'_\mathsf{f}, \mathsf{vk}'_\mathsf{f})$, obtains a signature of $f|\mathsf{vk_f}$ from its signing oracle $\sigma'' \leftarrow \mathsf{O_{Reg_{sig}}}(f|vk'_f)$, signs m using sk'_f himself, $\sigma' \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}'_\mathsf{f}, m)$, and outputs $(f(m), \sigma)$, where $\sigma = (m, c = (f, \mathsf{vk}'_\mathsf{f}, \sigma'), \sigma'')$.

If $\mathsf{A_{sig}}$ guessed correctly, eventually $\mathsf{A_{FS}}$ will output a **Type I** forgery, which must include a forgery with respect to to vk, and $\mathsf{A_{sig}}$ can use this forgery as its own forged signature in the unforgeability game for the standard signature scheme.

Case 2: b = 0: $\mathsf{A_{sig}}$ guesses that $\mathsf{A_{FS}}$ will produce a **Type II** forgery:

- $\mathsf{A_{sig}}$ generated a new key pair $(\mathsf{msk}, mathsfmvm)$ himself, and forwards mvk to $\mathsf{A_{FS}}$.
- when $\mathsf{A_{FS}}$ makes a $\mathsf{O_{key}}$ query for a function f, $\mathsf{A_{sig}}$ generates a new key pair $(\mathsf{sk}'_\mathsf{f}, \mathsf{vk}'_\mathsf{f}) \leftarrow \mathsf{Sig.Setup}(1^k)$, generates a signature $\sigma'' \leftarrow \mathsf{Sign}(\mathsf{msk}, f|\mathsf{vk}'_\mathsf{f})$ and outputs $\mathsf{sk_f} = (\mathsf{sk}'_\mathsf{f}, c = (f, \mathsf{vk}'_\mathsf{f}, \sigma''))$.
- to answer the signing queries for (f, m)
  - $\mathsf{A_{sig}}$ chooses a random $i \in [1, Q(k)]$ corresponding to the query in which he will embed the challenge.
  - for all signing queries other than the i^{th} one, $\mathsf{A_{sig}}$ chooses a new signing, verification key pair $(\mathsf{sk}'_\mathsf{f}, \mathsf{vk}'_\mathsf{f})$, generates a signature $\sigma'' \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f|vk'_f)$, and a signature $\sigma' \leftarrow \mathsf{Sign}(\mathsf{sk}'_\mathsf{f}, m)$, and outputs $\sigma = (f(m), (m, c = (f, \mathsf{vk}'_\mathsf{f}, \sigma''), \sigma')$.
  - $\mathsf{A_{sig}}$ plants his challenge verification key in the i^{th} query. It queries its oracle for a signature of m under vk, $\sigma' \leftarrow \mathsf{O_{Reg_{sig}}}(m)$, computes $\sigma'' \leftarrow \mathsf{Sig.Sign}(\mathsf{msk}, f|\mathsf{vk}))$, and outputs $(f(m), \sigma)$, where $\sigma = (m, c = (f, \mathsf{vk}, \sigma''), \sigma')$.

Eventually, if $\mathsf{A_{sig}}$ guessed correctly, $\mathsf{A_{FS}}$ will output a forgery that $\mathsf{A_{sig}}$ can use that as its forgery in the unforgeability game for the regular signature scheme.

$\mathsf{A_{sig}}$ is successful in the unforgeability game if:

- he guesses b correctly
- in the case that $b = 0$, he guesses the query i correctly

- A_{FS} outputs a forgery

His success probability is therefore:

$$\frac{1}{2}\frac{1}{Q(k)}\frac{1}{P(k)} = \frac{1}{2Q(k)P(k)}$$

This contradicts the unforgeability guarantee for the regular signature scheme, and therefore, assuming the original signature scheme satisfied unforgeability, the functional signature scheme in the construction above must also be unforgeable. □

While this construction is secure under very general assumptions (the existence of one-way functions), its efficiency can be greatly improved. The size of $\sigma \leftarrow \mathsf{FS.Sign}(sk_f, m)$ in this scheme is proportional to the size of $|f| + |m|$ plus the size of a signature of the standard signature scheme. This is in contrast to our SNARK-based construction, in which the signature size was proportional to $|f(m)|$, instead of $|f| + |m|$. In addition, the verification process is inefficient: the verifier has to compute $f(m)$ on its own. And, as mentioned before, it does not achieve any function privacy guarantees.

4 Applications of Functional Signatures

In this section we discuss applications of functional signatures to other cryptographic problems, such as constructing delegation scheme 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 is supposed to convince the verifier that an input x is in L. We require the proofs produced by prover to be sublinear in the size of the input plus the size of the witness.

We show how to use a functional signature that supports key for any function f, and has short signatures (i.e of size $\mathsf{poly}(k) \cdot o(|f(m)| + |m|)$) can be used to construct a SNARG scheme with preprocessing for any language $L \in NP$ with proof size $poly(k) \cdot o(|w| + |x|)$, where w is the witness and x is the instance.

Let L be an NP complete language, and R the corresponding relation. The main idea in the construction is for the verifier to give out a single signing key for a function whose range consists of exactly those strings that are in L. Then, with $\mathsf{sk_f}$, the prover will be able to sign only those messages that are in the language L and uses that as his proof. The proof is succinct and publicly verifiable. The construction is as follows:

- $\Pi.\mathsf{Gen}(1^k)$:
  - run the setup for the functional signature scheme, and get $(\mathsf{mvk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^k)$
  - 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 $\mathsf{FS.Sign}(\mathsf{sk_f}, x|w)$
- $\Pi.\mathsf{Verify}(\mathsf{crs}, x, \pi)$
  - output $\mathsf{FS.Verify}(\mathsf{mvk}, x, \pi)$

Theorem 32. *If* $(\mathsf{FE.Setup}, \Pi.\mathsf{Prove}, \mathsf{FS.Sign}, \mathsf{FS.Verify})$ *is a functional signature scheme,* $(\Pi.\mathsf{Gen}, \Pi.\mathsf{Prove}, \Pi.\mathsf{Verify})$ *is a succinct argument of knowledge.*

Correctness
The correctness property of the SNARG follows immediately from correctness property of the functional signature scheme.
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 that are in the range of f, and therefore in L.
Succinctness
The size of a proof is equal to the size of a signature in the functional signature scheme, $\mathsf{poly}(k) \cdot o(|f(m)| + |m|) = \mathsf{poly}(k) \cdot o(|x| + |w|)$.
We remark that Gentry and Wichs show in [GW11] that SNARG schemes with proof size $o(|w| + |x|)$ can not be obtained using black-box reductions to falsifiable assumptions, and therefore, in order to obtain a functional signature scheme with signature size $o(|f(m)| + |m|)$ we must either rely on non-falsifiable assumptions (as in our SNARK construction) or make use of non blackbox 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.

Given a functional signature scheme with with signature size $\delta(k)$, and verification time $t(k)$ we can get a delegation scheme in the preprocessing model with proof size $\delta(k)$ and verification time $t(k)$.
We construct a delegation scheme as follows:

- $\mathsf{KeyGen}(1^k, f)$:
  - run the setup for the functional signature scheme and generate $(\mathsf{mvm}, \mathsf{msk}) \leftarrow \mathsf{FS.Setup}(1^k)$.
  - let $f'(x) = (x, f(x))$, and get 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 $\mathsf{sk}_{f'} = \mathsf{evk}, x = \sigma_x$
  - get a signature of $(x, f(x))$, $\sigma \leftarrow \mathsf{FS.Sign}(\mathsf{sk}_{f'}, x)$
  - output $(f(x), \pi = \sigma)$
- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi_y)$:
  - output $\mathsf{FS.Verify}(\mathsf{vk}, y, \pi_y)$

Theorem 33. *If* $(\mathsf{FE.Setup}, \Pi.\mathsf{Prove}, \mathsf{FS.Sign}, \mathsf{FS.Verify})$ *is a functional signature scheme,* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Compute}, \mathsf{Verify})$ *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, the server will only be able to produce a signature of (x, y) that is in the range of f', that is if $y = f(x)$. So the server won't be able to sign a pair (x, y) with non-negligible probability, unless, $y = f(x)$.

5 Functional Pseudorandom Functions

In this section we present a formal definition and construction of functional pseudorandom functions (FPRF), pseudorandom functions with selective access (PRFSA), and hierarchical functional pseudorandom functions for function and predicate classes. 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*, based on the Goldreich-Goldwasser-Micali (GGM) tree-based PRF construction [GGM86]. This 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, knowledge of evaluation is either 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 (FPRF)* 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 34 (Functional PRF)**.** We say that a PRF family $\mathcal{F} = \{F_s : D \to R\}_{s \in S}$ is a *functional pseudorandom function (FPRF)* 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.

$\mathsf{Eval}(\mathsf{sk}_f, y)$: On input key sk_f and input $y \in D$, if it holds that there exists an $x \in A$ such that $f(x) = y$ then Eval outputs the PRF evaluation $F_s(y)$.

which satisfy the following properties:

- **Correctness:** For every (efficiently computable) function f, $\forall y \in D$ s.t. $\exists x$ for which $f(x) = 1$, it holds that
$$\forall s \leftarrow S,\ \forall sk_f \leftarrow \mathsf{KeyGen}(s, f),\ \mathsf{Eval}(\mathsf{sk}_f, y) = F_s(y).$$
- **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
$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$	$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$
$f_1 \leftarrow \mathcal{A}(\mathsf{pp})$	$f_1 \leftarrow \mathcal{A}(\mathsf{pp})$
$\mathsf{sk}_{f_1} \leftarrow \mathsf{KeyGen}(s, f_1)$	$\mathsf{sk}_P \leftarrow \mathsf{KeyGen}(s, f_1)$
$\vdots$	$\vdots$
$f_l \leftarrow \mathcal{A}(\mathsf{pp}, f_1, sk_{f_1}, \dots, f_{l-1}, sk_{f_{l-1}})$	$f_l \leftarrow \mathcal{A}(\mathsf{pp}, f_1, sk_{f_1}, \dots, f_{l-1}, sk_{f_{l-1}})$
$\mathsf{sk}_{f_l} \leftarrow \mathsf{KeyGen}(s, f_l)$	$\mathsf{sk}_{f_l} \leftarrow \mathsf{KeyGen}(s, f_l)$
Challenge Phase	Challenge Phase
$H \leftarrow \mathbb{F}_{D \to R}$	
$b \leftarrow \mathcal{A}^{\mathcal{O}^{\{f_i\}}_{s,H}(\cdot)}(f_1, \mathsf{sk}_{f_1}, \dots, f_l, \mathsf{sk}_{f_l})$	$b \leftarrow \mathcal{A}^{F_s(\cdot)}(f_1, \mathsf{sk}_{f_1}, \dots, f_l, \mathsf{sk}_{f_l})$

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

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 35 (Selectively Secure FPRF)**.** We say a PRF family is a *selectively secure* functional pseudorandom function if there exist additional algorithms $\mathsf{KeyGen}, \mathsf{Eval}$ satisfying the correctness property as above, in addition to 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
$H \leftarrow \mathbb{F}_{D \to R}$	
$b \leftarrow \mathcal{A}^{\mathcal{O}^{\{f_i\}}_{s,H}(\cdot)}(f_1, \mathsf{sk}_{f_1}, \ldots f_l, \mathsf{sk}_{f_l})$	$b \leftarrow \mathcal{A}^{F_s(\cdot)}(f_1, \mathsf{sk}_{f_1}, \ldots f_l, \mathsf{sk}_{f_l})$

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

Definition 36 (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 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.

$\mathsf{Eval}(\mathsf{sk}_P, x)$: 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

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

- **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
$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$	$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$
$P_1 \leftarrow \mathcal{A}(\mathsf{pp})$	$P \leftarrow \mathcal{A}(\mathsf{pp})$
$\mathsf{sk}_{P_1} \leftarrow \mathsf{KeyGen}(s, P_1)$	$\mathsf{sk}_P \leftarrow \mathsf{KeyGen}(s, P)$
$\vdots$	$\vdots$
$P_l \leftarrow \mathcal{A}(\mathsf{pp}, P_1, sk_{P_1} \ldots P_{l-1}, sk_{P_{l-1}})$	$P_l \leftarrow \mathcal{A}(\mathsf{pp}, P_1, sk_{P_1} \ldots P_{l-1}, sk_{P_{l-1}})$
$\mathsf{sk}_{P_l} \leftarrow \mathsf{KeyGen}(s, P_l)$	$\mathsf{sk}_{P_l} \leftarrow \mathsf{KeyGen}(s, P_l)$
Challenge Phase	Challenge Phase
$H \leftarrow \mathbb{F}_{D \to R}$	$s \leftarrow S$
$b \leftarrow \mathcal{A}^{\mathcal{O}^{P}_{s,H}(\cdot)}(P_1, \mathsf{sk}_{P_1}, \ldots P_l, \mathsf{sk}_{P_l})$	$b \leftarrow \mathcal{A}^{F_s(\cdot)}(P_1, \mathsf{sk}_{P_1}, \ldots P_l, \mathsf{sk}_{P_l})$

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

We also consider PRFSA that satisfy the selective pseudo randomness requirement.

Definition 37 (Selectively Secure PRFSA)**.** We say a PRF family with selective access is a *selectively secure* pseudorandom function with selective access if there exist additional algorithms $\mathsf{KeyGen}, \mathsf{Eval}$ satisfying the correctness property as above, in addition to 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 Rand	Experiment PRand
Query Phase	Query Phase
$P_1, \ldots, P_l \leftarrow \mathcal{A}$	$P_1, \ldots, P_l \leftarrow \mathcal{A}$
$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$	$(\mathsf{pp}, s) \leftarrow \mathsf{Gen}(1^k)$
$\mathsf{sk}_{P_1} \ldots \mathsf{sk}_{P_l} \leftarrow \mathsf{KeyGen}(s, P_1, \ldots P_l)$	$\mathsf{sk}_{P_1} \ldots \mathsf{sk}_{P_l} \leftarrow \mathsf{KeyGen}(s, P_1, \ldots P_l)$
Challenge Phase	Challenge Phase
$H \leftarrow \mathbb{F}_{D \to R}$	
$b \leftarrow \mathcal{A}^{\mathcal{O}^P_{s,H}(\cdot)}(P_1, \mathsf{sk}_{P_1}, \ldots P_l, \mathsf{sk}_{P_l})$	$b \leftarrow \mathcal{A}^{F_s(\cdot)}(P_1, \mathsf{sk}_{P_1}, \ldots P_l, \mathsf{sk}_{P_l})$

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

Definition 38 (Hierarchical FPRF)**.** We say that an FPRF 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 $F_s(y)$ on each value y for which there exists a preimage x with $g(x) = y$. Formally, for every sequence of (efficiently computable) functions $f_1, \ldots, f_\ell$ with $f_i : A_i \to A_{i-1}$, $\forall y \in A$ s.t. $\exists x$ for which $f_\ell \circ \cdots \circ f_1(x) = y$, it holds that

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

- **Pseudorandomness:** The pseudorandomness property of Definition 34 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 $id = 1$.

  $\mathsf{HonestKey}(id, g)$: 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 $\mathsf{sk}_{g \circ f} \leftarrow \mathsf{SubkeyGen}(\mathsf{sk}_f, g)$, and assigns this key a unique identity id'. The resulting key $\mathsf{sk}_{g \circ f}$ is kept secret.

  $\mathsf{CorruptKey}(id, g)$: 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 $\mathsf{sk}_{g \circ f} \leftarrow \mathsf{SubkeyGen}(\mathsf{sk}_f, g)$, and assigns this key a unique identity id'. The resulting key $\mathsf{sk}_{g \circ f}$ is given to the adversary.

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 $\mathsf{CorruptKey}$ 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, the class of functions our construction supports is

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

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, at level $|z|$. 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 $m = |z|$

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

Theorem 39. *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 35.*

Proof. We will reduce the pseudorandom property of our functional PRF scheme to the security of the underlying PRG. 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:

- In Exp I, in the key query phase, the key for each function f_i corresponding to prefix P_i is obtained by following the corresponding path in the GGM tree. In the challenge phase, the adversary's evaluation queries are answered with the corresponding pseudorandom values. This is exactly the experiment Sel-PRand in Definition 35.
- In Exp II, in the key query phase, the key for each function f_i corresponding to prefix P_i is computed as follows:
  - if no prefix of P_i is also queried by the adversary as one of the P_js, 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 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 (unique) evaluation queries are answered with random values.
- In Exp III, in the key query phase, the key for each function f_i corresponding to prefix P_i is obtained by following the corresponding path in GGM tree, and in the challenge phase the adversary's (unique) evaluation queries are answered with random values. This is the experiment Sel-Rand in Definition 35.

Note that Exp I is Experiment Sel-PRand in the Functional PRF definition, and Exp III is Experiment Sel-Rand. We will show that they are both computationally indistinguishable from Exp II as defined above, and therefore computationally indistinguishable from each other.

Suppose there exists an adversary $\mathsf{A}_{\mathsf{PRF}}$, who can distinguish between Exp I and Exp II with non-negligible advantage $\epsilon(n)$. We claim that we can use him to construct an adversary A_{PRG} that breaks the security of the underlying pseudorandom generator. The input to A_{PRG} is 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 $i \in [1, n]$. The value i corresponds to the level of the tree where A_{PRG} will place his challenge values when interacting with $\mathsf{A}_{\mathsf{PRF}}$.

In Exp^i, in the key query phase, the key for each function f_i 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 this string, as determined by the $(i+1)$ to n bits of the queried input x.

Since $\mathsf{A_{PRF}}$ can distinguish between Exp I and Exp II with probability $\epsilon(n)$, there must exist an i for which $\mathsf{A_{PRF}}$ distinguishes between Exp^i and Exp^{i+1} with probability $\frac{\epsilon(n)}{n}$.

Our constructed PRG adversary $\mathsf{A_{PRG}}$ plays the role of the challenger in the game with $\mathsf{A_{PRF}}$, chooses a random $i \in [1, n]$ 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 k^{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 k^{th} level of the tree.

In the challenge phase, the answers to the adversary's evaluation queries are computed as follows:
- at each point in which Exp^i would fill an ith-level node with a random string, one of $\mathsf{A_{PRG}}$'s challenge values is embedded in its place; then the PRGs are applied to this value as determined by the $(i+1)$ to n bits of the input.

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

At the end A_{PRG} outputs the same answer as $\mathsf{A_{PRF}}$ in its own security game.

A_{PRG} will be correct if it guessed i correctly, and if $\mathsf{A_{PRF}}$ distinguishes between Exp^i and Exp^{i+1}. A_{PRG} will therefore distinguish between random values and outputs of a pseudorandom generator with probability $\frac{\epsilon(n)}{n^2}$, which is non-negligible. This completes the proof that Exp I and Exp II are computationally indistinguishable.

We can use a very similar hybrid argument to show that Exp III and Exp II are computationally indistinguishable: 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 gets random values.

Then Exp I and Exp II, and Exp II and Exp III are computationally indistinguishable, Exp I and Exp III are also computationally indistinguishable, and therefore our construction satisfies the selective pseudo randomness definition for functionals PRFs.

□

Remark 40. We remark that one can directly obtain a *fully* secure FPRF for $\mathcal{F}_{\mathsf{pre}}$ (as in Definition 34) 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 $f_z \in \mathcal{F}_{\mathsf{pre}}$.

As an immediate corollary of Theorem 39, we obtain a *PRF with selective access* for the class of 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 41. *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 $\mathcal{P}_{\mathsf{pre}}$, as per Definition 37.*

Our FPRF 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 42. *Based on the existence of one-way functions, the GGM pseudorandom function family together with algorithms* KeyGen *and* Eval *defined as above, is a* hierarchical *functional PRF for the class of predicates $\mathcal{P}_{\mathsf{pre}}$.*

The fact the our construction satisfies the correctness property for hierarchical functional PRFs follows from the definition of the GGM pseudorandom function family.

The pseudorandomness property can be easily proved using the same techniques as in the proof of Theorem 39.

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.

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

[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.

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

[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.

[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.

[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.

[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.