Functional Signatures

by

Ioana Ivan

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author .....
Department of Electrical Engineering and Computer Science
May 22, 2013

Certified by 5/22/13
Shafrira Goldwasser
RSA Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski, Professor of Electrical Engineering
Chair of the Committee on Graduate Students

Functional Signatures

by

Ioana Ivan

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2013, in partial fulfillment of the
requirements for the degree of
Masters of Science in Computer Science and Engineering

Abstract

In this thesis, we introduce the notion of *functional signatures*. 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. An immediate application of functional signature scheme is the delegation by a master authority to a third party of the ability to sign a restricted set of messages. We also show applications of functional signature in constructing succinct non-interactive arguments and delegation schemes.

We give several constructions for this primitive, and describe the trade-offs between them in terms of the assumptions they require and the size of the signatures.

Thesis Supervisor: Shafrira Goldwasser
Title: RSA Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my advisor, Shafi Goldwasser for suggesting this problem, and for all her advice and encouragement. I am very grateful to be her student and to have the opportunity to learn from her.

This thesis is based on joint work with Elette Boyle. I want to thank her for many stimulating discussions and for reviewing a draft of this thesis.

Finally, I'd like to thank my family and friends for their support.

Contents

Chapter 1

Introduction

A digital signature scheme is a cryptographic primitive used for authenticating information. A signature on a message gives the receiver a way to verify that the message has been created by a proclaimed sender, and has not been modified by anyone else. The sender has a secret key, used in the signing process, and there is a corresponding verification key, which is public and can be used by the receiver to verify that a signature is valid. In [13], Goldwasser et al. formalized the generally adopted security requirement we require a signature scheme to satisfy, unforgeability against chosen message attack. Namely, 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 thesis, we introduce the notion of *functional signatures.* In a functional signature scheme, in addition to a signing key that can be used to sign any message(*the master signing key*), there are *signing keys for a function* 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 such a signature scheme should satisfy is that any probabilistic polynomial time (PPT) adversary, who can request signing keys for functions $f_1 \ldots f_l$ of his choice, and signatures for messages $m_1, \ldots m_q$ of his choice, can only produce a signature of a message m with non-negligible probability, if m is equal to one of the messages $m_1, \ldots m_q$, or if m is in the range of one of the functions $f_1 \ldots f_l$.

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

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

.

P could also be a predicate that checks if the message does not contain some phrases, or if i i's related to a certain subject, or if it satisfies a more complex policy.

Another 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 $\{f_i\}$'s 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.

An additional property one might desire 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 delegation of the signing algorithm context, the master authority might not wish to reveal which set of messages the assistant is allowed to sign. 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.

1.1 Summary of Our Results

We provide three constructions of functional signatures with various tradeoffs in computational assumptions and achieved security/efficiency.

Theorem 1 (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.*

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 [15].

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 .

There are two aspects of this construction that could be improved: the scheme doesn't satisfy function privacy, and the size of a signature output by the algorithm $\mathsf{Sign}(\mathsf{sk}_f, m)$ depends on the size of a circuit computing f. Ideally, we would want to signature to only depend on the size of the output $f(m)$ and the security parameter(or just the security parameter), and to hide both f and m.

Theorem 2 (Informal). *Assuming the existence of non-interactive zero knowledge arguments of knowledge (NIZKPoK) 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. The size of the signature is still dependent on the size of f and m.

Overview of the construction:

A (NIZKPoK) argument system for an NP language L with corresponding relation R, consists of a prover, P, and verifier V, that share a common reference string. The prover, who has a witness w such that $R(x, w) = 1$, can output a proof that convinces V that $x \in L$. On the other hand, the soundness property guarantees that if x is not in L, no polynomial time prover can output a proof that would make the verifier accept with non-negligible probability. For our construction, we actually need our proof system to be "proof of knowledge" which informally means that, if there exists a prover that can convince the verifier that $x \in L$, we can use this prover to extract a witness w such that $R(x, w) = 1$. These requirements are formalized in Chapter 2.

The setup algorithm for the functional signature scheme is the same as in the previous 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.

In the key generation algorithm for a function f, we just 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 NIZKPoK 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 NIZKPoK proof system.

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

Theorem 3 (Informal). *Assuming the existence of succinct non interactive arguments of knowledge (SNARKs) and NIZKPoK for NP languages, 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. The size of the signature only depends on the security parameter and the size of $f(m)$.

Overview of the construction:
A SNARK system for an NP language L with corresponding relation R is a proof system where the size of a proof is sublinear in the size of the witness corresponding to an instance. The running time of the verifier is required to be sublinear in the running time of R. SNARK schemes have been constructed under various non-falsifiable assumptions. For example, Bitansky et al. [3] construct SNARKs where the length of the proof and the verifiers 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. More details are given in Chapter 2.

To get a functional signature scheme where the size of a signature is independent of $|f|$ and $|m|$, we modify our NIZK-based construction as follows: instead of using a NIZK argument of knowledge, we use a zero-knowledge SNARK.

1.2 Related Work

1.2.1 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([14], [11], [12]). 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 [11], 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 is improved in a follow up work([12]), 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 [1], 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.

1.2.2 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 [2] 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:

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

[2] 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 [10], 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 [10] 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.

1.2.3 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 [8], 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([6]) only allows the evaluation of multivariate polynomials on signed data.
Constructing homomorphic signature schemes for general functions remains an open problem.

1.3 Overview of the Thesis

In **Chapter 2**, we describe several primitives which will be used in our constructions.
In **Chapter 3**, we give a formal definition of functional signature schemes, and present three constructions satisfying the definition.
In **Chapter 4**, we show how to construct delegation schemes and succinct non-interactive arguments (SNARGs) from functional signatures schemes.

Chapter 2

Preliminaries

In this chapter we define several cryptographic primitives that are used in our constructions of functional signatures.

2.1 Signature Schemes

Definition 4. A signature scheme for a message space $\mathcal{M}$ is a tuple $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$:

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

A signature scheme should satisfy the following properties:

Correctness

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

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 5 ([15]). *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 6. [7, 4, 5]: $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}))$ is an *efficient adaptive NIZK proof 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 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 $\mathcal{A}$,

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

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 $\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}(k)$ 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 proof of knowledge.

Definition 7. Let $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}))$ be an efficient adaptive NIZK proof system for an NP language $L \in \mathsf{NP}$ with a corresponding NP relation $\mathcal{R}$. We say that Π is a *proof-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 $\mathcal{A}$,

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

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 proof system Π to a NIZK proof-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 proof 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 8 ([7]). *Assuming the existence of enhanced trapdoor permutations, there exists an efficient adaptive NIZK proof of knowledge for all languages in NP.*

2.3 Succinct Non-Interactive Arguments (SNARGs)

Definition 9. $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ 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 $|\pi| = poly(k) \cdot o(|x| + |w|)$.

Definition 10. 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 it also satisfies the following property: there exists a PPT algorithm E such that for every PPT adversary $\mathcal{A}$,

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

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

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 [3] Bitansky et al. give a construction of SNARKs assuming the existence of extractable collision-resistant hash functions

Lemma 11 ([3]). *If there exist ECRHs and then there exist SNARKs for all languages in NP.*

In [9] 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 12. A *delegation scheme* for a function F consists of a tuple of algorithms (KeyGen, Encode, Compute, Verify)

- $\mathsf{KeyGen}(1^k, F) \rightarrow (\mathsf{enc}, \mathsf{evk}, \mathsf{vk})$: The key generation algorithm takes as input a security parameter k and a function F, and outputs a key enc that is used to encode the input, an evaluation key evk that is used for the evaluation of the function F, and a verification key vk that is used to verify that the output was computed correctly.

- $\mathsf{Encode}(\mathsf{enc}, x) \rightarrow \sigma_x$: The encoding algorithm uses the encoding key enc to encode the function input x as a public value σ_x, which is given to the server to compute with.

- $\mathsf{Compute}(\mathsf{evk}, \sigma_x) \rightarrow (y, \pi_y)$: Using the public evaluation key, evk and the encoded input σ_x, the server computes the function output y = F(x), and a proof π_y that y is the correct output.

- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi_y) \rightarrow \{0, 1\}$: The verification algorithm checks the proof π_y and outputs 1(indicating that the proof is correct), or 0 otherwise.

We require a delegation scheme to satisfy the following requirements:

Correctness

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

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

Authentication

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

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

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,

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

Chapter 3

Functional Signatures: Definition and Constructions

3.1 Formal Definiton

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 13. A *functional signature scheme* for a message space $\mathcal{M}$ consists of algorithms (FS.Setup, FS.KeyGen, FS.Sign, FS.Verify):

- $\mathsf{FS.Setup}(1^k) \rightarrow (\mathsf{msk}, \mathsf{mvk})$: the setup algorithm takes as input the security parameter and outputs the master signing key and master verification key.
- $\mathsf{FS.KeyGen}(\mathsf{msk}, f) \rightarrow \mathsf{sk_f}$: 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.
- $\mathsf{FS.Sign}(\mathsf{sk_f}, m) \rightarrow (f(m), \sigma)$: 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 $f(m)$.
- $\mathsf{FS.Verify}(\mathsf{mvk}, m^*, \sigma) \rightarrow \{0, 1\}$: 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 $\mathsf{FS.Sign}(\mathsf{sk}_{\mathsf{f_b}}, m_b)$, where $\mathsf{sk}_{\mathsf{f_b}} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f_b)$.

- The adversary wins the game if he guesses the bit b correctly.

3.2 Certificate-based construction

In this section we give a construction of a functional signatures from a standard signature scheme (i.e. existentially unforgeable under chosen message attack). Our functional signature scheme satisfies the unforgeability property given in Definition 13, but not function privacy. Since we can build standard signature schemes based on one-way functions (OWF) [15], 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 gotten 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 as functional signature scheme $(\mathsf{FS1.Setup}, \mathsf{FS1.KeyGen}, \mathsf{FS1.Sign}, \mathsf{FS1.Verify})$ as follows:

- $\mathsf{FS1.Setup}(1^k)$:

  - Sample a signing and verification key pair for the standard signature scheme $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Sig.Setup}(1^k)$, and set the master signing key to be msk, and the master verification key to be mvk.

- $\mathsf{FS1.KeyGen}(\mathsf{msk}, f)$:

- choose a new signing and verification key pair for the 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 $\mathsf{sk}_\mathsf{f} = (\mathsf{sk}'_\mathsf{f}, c)$.

- $\mathsf{FS1.Sign}(\mathsf{sk}_\mathsf{f}, m)$:
  - 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: $\mathsf{Sig.Sign}(\mathsf{sk}'_\mathsf{f}, m) \rightarrow \sigma'$.
  - let $\sigma = (m, c, \sigma')$
  - output $(f(m), \sigma)$

- $\mathsf{FS1.Verify}(\mathsf{mvk}, m^*, \sigma)$:
  - 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.

While this construction is secure under very general assumptions(the existence of one way functions), its efficiency can be greatly improved. The size of $\sigma \leftarrow FS.Sign(sk_f, m)$ is proportional to the size of $|f| + |m|$ plus the size of a signature of the standard signature scheme. Ideally, we would want the signature size to be proportional to $|f(m)|$, instead of $|f| + |m|$. The verification process is inefficient: the verifier has to compute $f(m)$ on its own, and, as mentioned before, we don't achieve any function privacy guarantees.

Our next construction satisfies both unforgeability and function privacy, although the size of a signature and the verification time are still not optimal.

Theorem 14. *If the signature scheme (*Sig.Setup, Sig.Sign, Sig.Verify*) is existentially unforgeable under chosen message attack, the functional signature scheme (*FS1.Setup, FS1.KeyGen, FS1.Sign, FS1.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. □

3.3 NIZK based construction

In order to get a functional signature scheme that also satisfies the function privacy requirement, we modify the previous construction to use non-interactive zero-knowledge proofs of knowledge (NIZKPoK). We remark that our construction hides the function f, but it reveals the size of a circuit computing f.

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 NIZK proof of knowledge 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 NIZKPOK Π, we'll construct a functional signature scheme $(\mathsf{FS2.Setup}, \mathsf{FS2.Keygen}, \mathsf{FS2.Sign}, \mathsf{FS2.Verify})$ as follows:

- $\mathsf{FS2.Setup}(1^k)$:
  - 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 NIZKPOK, $\mathsf{crs} \leftarrow \Pi.\mathsf{Gen}(1^k)$.
  - set the master secret key $\mathsf{msk} = \mathsf{vk}$, and the master verification key $\mathsf{mvk} = (\mathsf{vk}, \mathsf{crs})$.
- $\mathsf{FS2.KeyGen}(\mathsf{msk}, f)$:
  - 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 $\mathsf{sk}_f = c$

- FS2.Sign(sk_f, m):
  - let $\pi = \Pi.\mathsf{Prove}((f(m), \mathsf{mvk}), (f, m, \mathsf{sk_f}), \mathsf{crs})$ be a NIZKPOK 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 $(m^* = f(m), \sigma = \pi)$
- FS2.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 15. *If the signature scheme (*Sig.Setup, Sig.Sign, 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 non-interactive zero knowledge proof of knowledge, the functional signature scheme (*FS2.Setup, FS2.KeyGen, FS2.Sign, FS2.Verify*) as specified above satisfies both the unforgeability requirement and the function privacy requirement 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. Assuming the NIZKPOK system Π satisfies the extraction requirement in Definition 7, we show how to construct an adversary $\mathsf{A_{sig}}$ that produces 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 $\mathsf{O_{Reg_{sig}}}$. He is considered to be successful in producing a forgery if he outputs a valid signature for a message that was not queried from $\mathsf{O_{Reg_{sig}}}$.

$\mathsf{A_{sig}}$ interacts with $\mathsf{A_{FS}}$, playing the role of the challenger in the security game for the functional signature scheme. $\mathsf{A_{sig}}$ generates $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)$, a *simulated* crs for the NIZKPoK, together with a trapdoor, and forwards $(\mathsf{vk}, \mathsf{crs})$ as the master

verification key in the functional signature scheme to $\mathsf{A_{FS}}$.

Note that by the perfect extraction property, we can guaranteed that the probability of $\mathsf{A_{FS}}$ producing a forgery when given the simulated crs can only decrease by a negligible amount.

$\mathsf{A_{FS}}$ makes two types of queries:

- $\mathsf{O_{key}}(f)$, which $\mathsf{A_{sig}}$ answers by forwarding f to its signing oracle.
- $\mathsf{O_{sign}}(f, m)$, 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 $\sigma \leftarrow \mathsf{O_{Reg_{sig}}}(f')$. 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 signature is generated differently than in FS.Sign algorithm, but, because of the zero knowledge property of the proof system Π, we can guaranteed that the probability of $\mathsf{A_{FS}}$ producing a forgery can only decrease by a negligible amount.

After quering the oracles, $\mathsf{A_{FS}}$ will output a proof π^* that $(m^*, \mathsf{vk}) \in L$ for an m^* such that:

- there is no f that was a query to $\mathsf{O_{key}}$ such that $m^* = f(m)$ for some m.
- there is no (f, m) tuple that was a query to $\mathsf{O_{sign}}$ such that $m^* = f(m)$

$\mathsf{A_{sig}}$ can run $\mathsf{E}_2(\mathsf{crs}, \mathsf{trap}, (\mathsf{m}^*, \mathsf{vk}), \pi)$ to recover a witness $w = (f, m, \sigma)$ such that $m^* = f(m)$ and $\mathsf{Sig.Verify}(\mathsf{vk}, f, \sigma) = 1$.

$\mathsf{A_{sig}}$ can use σ as a forgery in the unforgettability game for the regular signature scheme.

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.

This adversary can break the zero-knowledge property of the proof system, since it can distinguish between proofs generated using different witnesses.

□

3.4 Construction based on SNARKS

In this section, we discuss a functional signature scheme that is secure under less standard assumptions, but greatly improves some parameters of the previous two schemes: the size of a signature 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 16 ([3]). *If there exist SNARKs, there exist zero-knowledge SNARKs.*

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

Theorem 17. *If (Sig.Setup, Sig.Sign, Sig.Verify) is an existentially unforgeable signature scheme, and Π' is a SNARK, our new functional signature construction (*FS3.Setup, FS3.Keygen, FS3.Sign, FS3.Verify*) satisfies both unforgeability and function privacy.*

We can use the proof from the previous section, since a zero-knowledge SNARK and a NIZK satisfy the same 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.

Chapter 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 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 18. *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 [9] 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 19. *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, $y = f(x)$.

Bibliography

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

[2] 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.

[3] 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.

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

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

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

[7] 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.

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

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

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

[11] 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.

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

[13] 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.

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

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