A Punctured Programming Approach to Adaptively Secure Functional Encryption

Brent Waters[⋆]

University of Texas at Austin, bwaters@cs.utexas.edu

Abstract. We propose the first construction for achieving adaptively secure functional encryption (FE) for poly-sized circuits (without complexity leveraging) from indistinguishability obfuscation ($i\mathcal{O}$). Our reduction has polynomial loss to the underlying primitives. We develop a "punctured programming" approach to constructing and proving systems where outside of obfuscation we rely only on primitives realizable from pseudo random generators.

Our work consists of two constructions. Our first FE construction is provably secure against any attacker that is limited to making all of its private key queries *after* it sees the challenge ciphertext. (This notion implies selective security.) Our construction makes use of an we introduce called puncturable deterministic encryption (PDE) which may be of independent interest. With this primitive in place we show a simple FE construction.

We then provide a second construction that achieves adaptive security from indistinguishability obfuscation. Our central idea is to achieve an adaptively secure functional encryption by bootstrapping from a one-bounded FE scheme that is adaptively secure. By using bootstrapping we can use"selective-ish" techniques at the outer level obfuscation level and push down the challenge of dealing with adaptive security to the one-bounded FE scheme, where it has been already been solved. We combine our bootstrapping framework with a new "key signaling" technique to achieve our construction and proof. Altogether, we achieve the first construction and proof for adaptive security for functional encryption.

1 Introduction

In traditional encryption systems a message, m, is encrypted with a particular user's public key PK. Later a user that holds the corresponding secret key will be able to decrypt the ciphertext and learn the contents of the message. At the same time any computationally bounded attacker will be unable to get any additional information on the message.

While this communication paradigm is appropriate for many scenarios such as targeted sharing between users, there exist many applications that demand a

[⋆] Brent Waters is supported by NSF CNS-0915361 and CNS-0952692, CNS-1228599 DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, DARPA N11AP20006, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

more nuanced approach to sharing encrypted data. For example, suppose that an organization encrypts video surveillance images and stores these ciphertexts in a large online database. Later, we would like to give an analyst the ability to view all images that match a particular pattern such as ones that include a facial image that pattern matches with a particular individual. In a traditional encryptions system we would be forced to either give the analyst the secret key enabling them to view everything or give them nothing and no help at all.

The concept of functional encryption (FE) was proposed to move beyond this all or nothing view of decryption. In a functional encryption system a secret key SK_f is associated with a function f. When a user attempts to decrypt a ciphertext CT encrypted for message m with secret key SK_f, he will learn $f(m)$. The security of functional encryption states that an attacker that receives keys for any polynomial number of functions $f_1, \ldots, f_Q$ should not be able to distinguish between an encryption of m_0, m_1 as long as $\forall i \; f_i(m_0) = f_i(m_1)$.

The concept of functional encryption first appeared under the guise of predicate encryption [BW07,KSW08] with the nomenclature later being updated [SW08,BSW11] to functional encryption. In addition, functional encryption has early roots in Attribute-Based Encryption [SW05] and searching on encrypted data [BCOP04].

A central challenge is to achieve functional encryption for as expressive functionality classes as possible — ideally one would like to achieve it for any poly-time computable function. Until recently, the best available was roughly limited to the inner product functionality proposed by Katz, Sahai, and Waters [KSW08]. This state of affairs changed dramatically with the introduction of a candidate indistinguishability obfuscation [BGI+12] system for all poly-size circuits by Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH+13] (GGHRSW). The authors showed that a functional encryption system for any poly-sized circuits can be built from an indistinguishability obfuscator plus public key encryption and statistically simulation sound non-interactive zero knowledge proofs.

Thinking of Adaptive Security While the jump from inner product functionality to any poly-size circuit is quite significant, one limitation of the GGHRSW functional encryption system is that it only offers a *selective* proof of security where the attacker must declare the challenge messages before seeing the parameters of the FE system. Subsequently, Boyle, Chung and Pass [BCP14] proposed an FE construction based on an obfuscator that is differing inputs secure. We briefly recall that an obfuscator $\mathcal{O}$ is indistinguishability secure if it is computationally difficult for an attacker to distinguish between obfuscations $\mathcal{O}(C_0)$ and $\mathcal{O}(C_1)$ for any two (similar sized) circuits that are functionally equivalent (i.e. $\forall x \; C_0(x) = C_1(x)$). Recall, that differing inputs [BCP14,ABG+13] security allows for an attacker to use circuits C_0 and C_1 that are not functionally equivalent, but requires that for any PPT attacker that distinguishes between obfuscations of the two circuits there must a PPT extraction algorithm that finds some x such that $C_0(x) \neq C_1(x)$. Thus, differing inputs obfuscation is in a qualitatively different class of "knowledge definitions". Furthermore, there is significant evidence [GGHW14] that there exist certain functionalities with auxiliary input that are impossible to build obfuscate under the differing inputs definition.

Our goal is to build adaptively secure functional encryption systems from indistinguishability obfuscation. We require that our reductions have polynomial loss of security relative to the underlying primitives. (In particular, we want to avoid the folklore complexity leveraging transformation of simply guessing the challenge messages with an exponential loss.) In addition, we want to take a minimalist approach to the primitives we utilize outside of obfuscation. In particular, we wish to avoid the use of additional "strong tools" such as non-interactive zero knowledge proofs or additional assumptions over algebraic groups. We note that our focus is on indistinguishability notions of functional encryption as opposed to simulation definitions [BSW11,O'N10].

Our Results In this work we propose two new constructions for achieving secure functional encryption (for poly-sized circuits) from indistinguishability obfuscation. We develop a "punctured programming" approach [SW14] to constructing and proving systems where our main tools in addition to obfuscation are a selectively secure puncturable pseudo random functions. We emphasize puncturable PRFs are themselves constructible from pseudo random generators [GGM84,BW13,BGI13,KPTZ13].

We start toward our FE construction which is provably secure against any attacker that is limited to making all of its private key queries *after* it sees the challenge ciphertext.[1] While this is attacker is still restricted relative to a fully adaptive attacker, we observe that such a definition is already stronger than the commonly used selective restriction.

To build our system we first introduce an abstraction that we call puncturable deterministic encryption (PDE). The main purpose of this abstraction is to serve in some places as a slightly higher level and more convenient abstraction to work with than puncturable PRFs. A PDE system is a symmetric key and deterministic encryption scheme and consists of four algorithms: $\mathsf{Setup}_{\mathsf{PDE}}(1^\lambda)$, $\mathsf{Encrypt}_{\mathsf{PDE}}(K, m)$, $\mathsf{Decrypt}_{\mathsf{PDE}}(K, \mathrm{CT})$, and $\mathsf{Puncture}_{\mathsf{PDE}}(K, m_0, m_1)$. The first three algorithms have the usual correctness semantics. The fourth puncture algorithm takes as input a master key and two messages (m_0, m_1) and outputs a punctured key that can decrypt all ciphertexts except for those encrypted for either of the two messages — recall encryption is deterministic so there are only two such ciphertexts. The security property of PDE is stated as a game where the attacker gives two messages (m_0, m_1) to the attacker and then returns back a punctured key as well as two ciphertexts, one encrypted under each message. In a secure system no PPT attacker will be able to distinguish which ciphertext is associated with which message.

Our PDE encryption mechanism is rather simple and is derived from the hidden trigger mechanism from the Sahai-Waters [SW14] deniable encryption

[1] This model has been called semi-adaptive in other contexts [CW14].

scheme. PDE Ciphertexts are of the form:

$$\mathrm{CT} = (A = F_1(K_1, m), \quad B = F_2(K_2, A) \oplus m).$$

where F_1 and F_2 are puncturable pseudo random functions, with F_1 being an injective function. Decryption requires first computing $m' = B \oplus F_2(K_2, A)$ and then checking that $F_1(K_1, m') = A$. [2]

With this tool in place we are now ready to describe our first construction. The setup algorithm will first choose a puncturable PRF key K for function F. Next, it will create the public parameters PP as an obfuscation of a program called InitialEncrypt. The InitialEncrypt program will take in randomness r and compute a tag $t = \mathrm{PRG}(r)$. Then it will output t and a PDE key k that is derived from $F(K, t)$. The encryption algorithm can use this obfuscated program to encrypt as follows. It will simply choose a random value $r \in \{0, 1\}^\lambda$, where λ is the security parameter. It then runs the obfuscated program on r to receive (t, k) and then creates the ciphertext CT as $(t, c = \mathsf{Encrypt}_{\mathsf{PDE}}(k, m))$.

The secret key SK_f for a function f will be created as an obfuscated program. This program will take as input a ciphertext $\mathrm{CT} = (t, c)$. The program first computes k from $F(K, t)$, then uses k to decrypt c to a message m and outputs $f(m)$. The decryption algorithm is simply to run the obfuscated program on the ciphertext.

The proof of security of our first system follows what we can a "key-programming" approach. The high level idea is that for each key we will hardwire in the decryption response into each secret key obfuscated program for when the input is the challenge ciphertext. For all other inputs the key computes decryption normally. Our key-programming approach is enabled by two important factors. First, in the security game there is a single challenge ciphertext so only one hardwiring needs to be done per key. Second, since all queries come after the challenge messages (m_0, m_1) are declared we will know where we need to puncture to create our hardwiring.

Intuitively, our proof can be broken down into two high level steps. First, we will perform a set of steps that allow us to hardwire the decryption answers to all of the secret keys for the challenge ciphertext. Next, we use PDE security to move from encrypting m_b for challenge bit $b \in \{0, 1\}$ to always encrypting m_0— independent of the bit b. (The actual proof contains multiple hybrids and is more intricate.)

Handling Full Security We now move to dealing with full security where we need to handle private key queries on both sides of the challenge ciphertext. At this point it is clear that relying only on key-programming will not suffice. First, a pre-challenge ciphertext key for function f will need to be created before the challenge messages (m_0, m_1) are declared, so it will not even be known at key creation time what $f(m_0) = f(m_1)$ will be.

[2] Despite sharing the term deterministic, our security definition of PDEs does not have much in common with deterministic encryption [BFO08,BFOR08] which has a central goal of hiding information among message distributions of high entropy.

Our central idea is to achieve an adaptively secure functional encryption by bootstrapping from a one-bounded FE scheme that is adaptively secure. At a high level a ciphertext is associated with a tag t and a private key with a tag y. From the pair of tags (t, y) one can (with the proper key material) pseudorandomly derive a master secret key k for a one bounded FE system. The ciphertext will be equipped with an obfuscated program, C, which on input of a key tag y will generate the one bounded key k (associated with the pair (t, y)) and then uses this to create an encryption of the message m under the one-bounded scheme with key k. Likewise, the private key for functionality f comes equipped with an obfuscated program P_f which on input of a ciphertext tag t derives the one bounded secret key k and uses this to create a one-bounded secret key.

The decryption algorithm will pass the key tag y to the ciphertext program to get a one bounded ciphertext $\mathrm{CT}_{\mathrm{OB}}$ and the ciphertext tag t to the key program to get a one bound key $\mathrm{SK}_{\mathrm{OB}}$. Finally, it will apply the one bounded decryption algorithm as $\mathsf{DecryptOB}(\mathrm{CT}_{\mathrm{OB}}, \mathrm{SK}_{\mathrm{OB}})$ to learn the message m. The one bounded key and ciphertext are compatible since they are both derived psuedorandomly from the pair (t, y) to get *same* one-bounded key k. (Note a different pair $(t', y') \neq (t, y)$ corresponds to a different one bounded FE key k' with high probability.)

Our bootstrapping proof structure allows us to develop "selective-ish" techniques at the outer level since in our reductions the ciphertext and private key tags can be chosen randomly ahead of time before the challenge message or any private key queries are known. Then the challenge of dealing with adaptive security is then "pushed down" to the one bounded FE scheme, where it has been solved in previous work [GVW12].

In the description above we have so far omitted one critical ingredient. In addition to generating a one bounded secret key on input t, the program P_f on input t will also generate an encrypted signal a that is passed along with the tag y to the ciphertext program C on decryption to let it know that it is "okay" to generate the one-bounded ciphertext for the pair (t, y). In the actual use of the system, this is the only functionality of the signal. However, looking ahead to our proof we will change the signal encrypted to tell the program C to switch the message for which it generates one bounded encryption encryptions of.

Our proof replaces key programming with a method we call "key-signaling". In a key-signaling system a normal ciphertext will be associated with a single message m which we refer to as an α-message. The decryption algorithm will use the secret key to prepare an α-signal for the ciphertext which will enable normal decryption. However, the ciphertext can also have a second form in which it is associated with two messages m_α and m_β. The underlying semantics are that if it receives an α-signal it uses m_α and if it receives a β-signal it uses m_β.

These added semantics open up new strategy for proving security. In the initial security game the challenge ciphertext encrypts m_b for challenge bit b. It will only receive α-signals from keys. Next we (indistinguishably) move the challenge ciphertext to encrypt m_b as the α-message and m_0 as the β-message. All keys still send only α-signals. Now one by one we change each key to send an β-signal to the challenge ciphertext as opposed to an α-signal. This step is feasible since for any queried function f we must have that $f(m_b) = f(m_0)$. Finally, we are able to erase the message m_b since no key is signaling for it.

Stepping back we can see that instead of storing the response of decryption for the challenge ciphertext at each key, we are storing the fact that it is using the second message in decryption.

We note that we can instantiate the one-bounded system using the construction of Gorbunov, Vaikuntanathan and Wee [GVW12] (GVW) who proved adaptive security of a public key FE 1-bounded scheme from IND-CPA secure public key encryption. Since we actually only need master key encryption, we observe that this can be achieved from IND-CPA symmetric key encryption. Thus, we maintain our goal of not using heavy weight primitives outside of obfuscation. One important fact is that the GVW scheme is proven to be 1-bounded adaptively secure regardless of whether the private key query comes before or after the challenge ciphertext. We note that the GVW system actually allows for a single key, but many ciphertexts; however, we only require security for a single ciphertext. The actual proof of security requires several hybrid steps and we defer further details to Section 5.

Recent Work Recently, Garg, Gentry, Halevi, and Zhandry [GGHZ14a] showed how to realize adaptively secure Attribute-Based Encryption from multilinear graded encodings. It is based on $\mathcal{U}$-graded encodings.

Subsequent to both of these works, the same authors [GGHZ14b] gave a construction of Functional Encryption from multilinear encodings. This construction required a new multilinear encoding functionality of allowing the "encoding grades" to be dynamically extended by any party using just the public parameters. Their scheme crucially leverages this capability and is also reflected in the assumption.

There are different tradeoffs between and pure indistinguishability obfuscation approach and that used in [GGHZ14b]. On one hand the approach of [GGHZ14b] allows one to directly get to mutlilinear encodings. On the other hand the novel use of extensions of grades both gives a novel technical idea, but possibly presents new risks. For example, there has been a flurry of recent activity consisting of attacks and responses to certain candidate constructions and assumptions of multilinear enocdings [CHL$^+$14,BWZ14,GHMS14,CLT14].

If one reduces to indistinguishability obfuscation, it can potentially be realized from different types of assumptions, including different forms of multilinear encodings or potentially entirely different number theory. An interesting open question is whether indistinguishability obfuscation or some close variant of it can be reduced to a basic number theoretic assumption that does not rely on sub exponential hardness. One interesting variant of this direction is to consider different variations of $i\mathcal{O}$ that are more amenable to such proofs, but can be leveraged in similar ways.

Bootstrapping with a Flipped One-time FE Scheme More recently, Ananth, Brakerski, Segev and Vaikuntanathan [ABSV14] showed an eloquent adaptation of our technique of bootstrapping from an adaptive 1-bounded scheme. Instead of starting with the 1-bounded FE scheme of GVW, they use a simple transformation on GVW due Brakerski and Segev[BS14] and applying universal circuits to create a flipped version of it. While the GVW scheme we used can handle a single key and many ciphertexts, the flipped version does the opposite. It can handle multiple keys, but only generating one ciphertext (this is done with secret key encryption).

They go on to show that using the flipped version of one-bounded FE for bootstrapping enables simplifications in the construction and proof. Instead of having attaching an obfuscated program to the ciphertext to generate one-bounded ciphertexts, the composite ciphertext contains a single 1-bounded ciphertext. In addition, it has a separate ("trojan") component that allows for transmitting the 1-bounded secret key used create a ciphertext to a program on the key side. Taken together the flipping and the trojan transmission allow for the private key to consist of a selectively secure functional encryption system.

2 Functional Encryption

Definition 1 (Functional Encryption). *A* functional encryption scheme *for a class of functions $\mathcal{F} = \mathcal{F}(\lambda)$ over message space $\mathcal{M} = \mathcal{M}(\lambda)$ consists of four algorithms $\mathcal{FE} = \{\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}\}$:*

$\mathsf{Setup}(1^\lambda)$ *– a polynomial time algorithm that takes the unary representation of the security parameter λ and outputs public parameters* PP *and a master secret key* MSK*.*

$\mathsf{KeyGen}(\mathrm{MSK}, f)$ *– a polynomial time algorithm that takes as input the master secret key* MSK *and a description of function $f \in \mathcal{F}$ and outputs a corresponding secret key* SK_f*.*

$\mathsf{Encrypt}(\mathrm{PP}, x)$ *– a polynomial time algorithm that takes the public parameters* PP *and a string x and outputs a ciphertext* CT*.*

$\mathsf{Decrypt}(\mathrm{SK}_f, \mathrm{CT})$ *– a polynomial time algorithm that takes a secret key* SK_f *and ciphertext encrypting message $m \in \mathcal{M}$ and outputs $f(m)$.*

A functional encryption scheme is correct for $\mathcal{F}$ if for all $f \in \mathcal{F}$ and all messages $m \in \mathcal{M}$:

$$\Pr[\ (\mathrm{PP}, \mathrm{MSK}) \leftarrow \mathsf{Setup}(\mathbf{1}^\lambda);$$
$$\mathsf{Decrypt}(\mathsf{KeyGen}(\mathrm{MSK}, f), \mathsf{Encrypt}(\mathrm{PP}, m)) \neq f(m)\ ] = negl(\lambda).$$

Indistinguishability Security for Functional Encryption We describe indistinguishability security as a multi-phased game between an attacker $\mathcal{A}$ and a challenger.

Setup: The challengers runs $(\mathrm{PP}, \mathrm{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives PP to $\mathcal{A}$.

Query Phase 1: $\mathcal{A}$ adaptively submits queries f in $\mathcal{F}$ and is given $\mathrm{SK}_f \leftarrow \mathsf{KeyGen}(\mathrm{MSK}, f)$. This step can be repeated any polynomial number of times by the attacker.

Challenge: $\mathcal{A}$ submits two messages $m_0, m_1 \in \mathcal{M}$ such that $f(m_0) = f_(m_1)$ for all functions f queried in the key query phase. The challenger then samples $\mathrm{CT}^* \leftarrow \mathsf{Encrypt}(\mathrm{PP}, m_b)$ for the attacker.

Query Phase 2: $\mathcal{A}$ continues to issue key queries as before subject to the restriction that any f queried must satisfy $f(m_0) = f(m_1)$.

Guess: $\mathcal{A}$ eventually outputs a bit b' in $\{0, 1\}$.

The advantage of an algorithm $\mathcal{A}$ in this game is $\mathsf{Adv}_{\mathcal{A}} = \Pr[b' = b] - \frac{1}{2}$.

Definition 2. *A functional encryption scheme is indistinguishability secure if for all poly-time $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ is negligible.*

Definition 3. *In the above security game we define a* post challenge ciphertext *attacker as one that does not make any key queries in Phase 1. We define a functional encryption scheme to be* post challenge ciphertext *indistinguishability secure if for any poly-time algorithm $\mathcal{A}$ that is a post challenge ciphertext attacker the advantage of $\mathcal{A}$ is negligible in the indistinguishability security game.* [3]

3 Puncturable Deterministic Encryption

In this section we define a primitive of puncturable deterministic encryption and show how to build it from (injective) puncturable PRFs. The main purpose of this abstraction is to give a slightly higher level tool (relative to puncturable PRFs) to work with in our punctured programming construction and proofs.

Definition 4 (Puncturable Deterministic Encryption). *A* puncturable deterministic encryption *(PDE) scheme is defined over a message space $\mathcal{M} = \mathcal{M}(\lambda)$ and consists of four algorithms: (possibly) randomized algorithms* $\mathsf{Setup}_{\mathsf{PDE}}$*, and* $\mathsf{Puncture}_{\mathsf{PDE}}$ *along with deterministic algorithms* $\mathsf{Encrypt}_{\mathsf{PDE}}$ *and* $\mathsf{Decrypt}_{\mathsf{PDE}}$*. All algorithms will be poly-time in the security parameter.*

$\mathsf{Setup}_{\mathsf{PDE}}(1^\lambda)$ *The setup algorithm takes a security parameter and uses its random coins to generate a key K from a keyspace $\mathcal{K}$.*

$\mathsf{Encrypt}_{\mathsf{PDE}}(K, m)$ *The encrypt algorithm takes as input a key K and a message m. It outputs a ciphertext* CT. The algorithm is deterministic.

$\mathsf{Decrypt}_{\mathsf{PDE}}(K, \mathrm{CT})$ *The decrypt algorithm takes as input a key K and ciphertext* CT*. It outputs either a message $m \in \mathcal{M}$ or a special reject symbol $\perp$.*

$\mathsf{Puncture}_{\mathsf{PDE}}(K, m_0, m_1)$ *The puncture algorithm takes as input a key $K \in \mathcal{K}$ as well as two messages m_0, m_1. It creates and outputs a new key $K(m_0, m_1) \in \mathcal{K}$. The parentheses are used to syntactically indicate what is punctured.*

[3] We remark that any system that is *post challenge ciphertext* secure must also be selectively secure.

Correctness A punctured deterministic encryption scheme is correct if there exists a negligible function negl such that the following holds for all λ and all pairs of messages $m_0, m_1 \in \mathcal{M}(\lambda)$.

Let $K = \mathsf{Setup}_{\mathsf{PDE}}(1^\lambda)$ and $K(m_0, m_1) \leftarrow \mathsf{Puncture}_{\mathsf{PDE}}(K, m_0, m_1)$. Then for all $m \neq m_0, m_1$

$$\Pr[\mathsf{Decrypt}_{\mathsf{PDE}}(K(m_0, m_1), \mathsf{Encrypt}_{\mathsf{PDE}}(K, m)) \neq m] = \mathrm{negl}(\lambda).$$

In addition, we have that for all m (including m_0, m_1)

$$\Pr[\mathsf{Decrypt}_{\mathsf{PDE}}(K, \mathsf{Encrypt}_{\mathsf{PDE}}(K, m)) \neq m] = \mathrm{negl}(\lambda).$$

Definition 5. *We say that a correct scheme is perfectly correct if the above probability is 0 and otherwise say that it is statistically correct.*

(Selective) Indistinguishability Security for Punctured Deterministic Encryption We describe indistinguishability security as a multi-phased game between an attacker $\mathcal{A}$ and a challenger.

Setup: The attacker selects two messages $m_0, m_1 \in \mathcal{M}$ and sends these to the challenger. The challenger runs $K = \mathsf{Setup}_{\mathsf{PDE}}(1^\lambda)$ and $K(m_0, m_1) = \mathsf{Puncture}_{\mathsf{PDE}}(K, m_0, m_1)$. It then chooses a random bit $b \in \{0, 1\}$ and computes

$$T_0 = \mathsf{Encrypt}_{\mathsf{PDE}}(K, m_b),\ T_1 = \mathsf{Encrypt}_{\mathsf{PDE}}(K, m_{1-b}).$$

It gives the punctured key $K(m_0, m_1)$ as well as T_0, T_1 to the attacker.

Guess: $\mathcal{A}$ outputs a bit b' in $\{0, 1\}$.

The advantage of an algorithm $\mathcal{A}$ in this game is $\mathsf{Adv}_{\mathcal{A}} = \Pr[b' = b] - \frac{1}{2}$.

Definition 6. *A puncturable deterministic encryption scheme is indistinguishability secure if for all poly-time $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ is negligible.*

Sampling Master Keys At times instead of running the $\mathsf{Setup}_{\mathsf{PDE}}(1^\lambda)$ algorithm to generate the master key for a PDE scheme we will generate the master key by simply sampling a uniformly random string $k \in \{0, 1\}^\lambda$ where λ is the security parameter. We can also do this without loss of generality.

In our full version [Wat14] we give a construction of puncturable deterministic encryption puncturable PRFs. This follows the hidden triggers construction from [SW14].

4 A Post Challenge Ciphertext Secure Construction

We now describe our construction for a functional encryption (FE) scheme that is post challenge ciphertext secure. We let the message space $\mathcal{M} = \mathcal{M}(\lambda) = \{0, 1\}^{\ell(\lambda)}$ for some polynomial function ℓ and the function class be $\mathcal{F} = \mathcal{F}(\lambda)$.

We will use a puncturable PRF $F(\cdot,\cdot)$ such that when we fix the key K we have that $F(K,\cdot)$ takes in a 2λ bit input and outputs λ bits. In addition, we use a puncturable deterministic encryption scheme (PDE) where the message space $\mathcal{M}$ is the same as that of the (FE) system. In our PDE systems master (non-punctured) keys are sampled uniformly at random from $\{0,1\}^\lambda$. Finally, we use an indistinguishability secure obfuscator and a length doubling pseudo random generator PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda}$.

Our Construction In our system the setup algorithm will produce an obfuscated program P that serves as the public parameters. Encryption proceeds in two steps. First the encryptor will choose a random string r and run $P(r)$. The obfuscated program will first use r to generate a tag t. Next the program will apply a (puncturable) psuedorandom function on t with global key K to generate a PDE key k. The program outputs both the tag t and PDE key k to the encryptor. Finally, the encryptor will use k to perform an encryption of the actual message m getting PDE ciphertext c. The (total) ciphertext CT consists of the tag t and c. Intuitively, the ciphertext component c is the "core encryption" of the message and the tag t tells how one can derive the PDE key k (if one knows the system's puncturable PRF key).

The authority generates a private key for function f as an obfuscated program P_f. To decrypt a ciphertext $\mathrm{CT} = (t,c)$ the decrypt or simply runs $P_f(t,c)$. The obfuscated program will first generate *the same* PDE key k that was used to encrypt the ciphertext

We make two intuitive remarks about security. First, we note that the system's puncturable PRF key K only appears in obfuscated programs and not in the clear. Second, it is not necessarily a problem perform the core encryption of the message under a *deterministic* scheme. The reason is that the encryption procedure implicitly chooses a fresh k so with high probability any single PDE key should only be used once. (Clearly, performing a deterministic encryption step more than once with the same key would be problematic.)

We now give our construction in detail.

$\mathsf{Setup}(1^\lambda)$
The setup algorithm first chooses a random punctured PRF key $K \leftarrow \mathrm{Key}_F(1^\lambda)$ and sets this as the master secret key MSK. Next it creates an obfuscation of the program Initial-Encrypt as $P \leftarrow i\mathcal{O}(1^\lambda, \textsc{Initial-Encrypt}{:}1[K])$.[4] (See Figure 1.) This obfuscated program, P, serves as the public parameters PP.

$\mathsf{Encrypt}(\mathrm{PP} = P(\cdot), m \in \mathcal{M})$
The encryption algorithm chooses random $r \in \{0,1\}^\lambda$. It then runs the obfuscated program P on r to get:

$$(t,k) \leftarrow P(r).$$

It then computes $\mathsf{Encrypt}_{\mathsf{PDE}}(k,m) = c$. The output ciphertext is $\mathrm{CT} = (t,c)$.

[4] The program Initial-Encrypt:1 is padded to be the same size as Initial-Encrypt:2.

$\mathsf{KeyGen}(\text{MSK}, f \in \mathcal{F}(\lambda))$ The KeyGen algorithm produces an obfuscated program P_f by obfuscating

$$P_f \leftarrow i\mathcal{O}(\text{Key-Eval:1}[K, f]).^5$$

$\mathsf{Decrypt}(\text{CT} = (t, c), \text{SK} = P_f)$ The decryption algorithm takes as input a ciphertext CT and a secret key SK which is an obfuscated program P_f. It runs $P_f(t, c)$ and outputs the response.

Correctness Correctness follows in a rather straightforward manner from the correctness of the underlying primitives. We briefly sketch the correctness argument. Suppose we call the encryption algorithm for message m with randomness r. The obfuscated program generates $(t, k) = (\text{PRG}(r), F(K, t))$. Then it creates the ciphertext $\text{CT} = (t, c = \mathsf{Encrypt}_{\mathsf{PDE}}(k, m))$. Now let's examine what occurs when $\mathsf{Decrypt}(\text{CT} = (t, c), \text{SK}_f = P_f)$ is called where P_f was a secret key created from function f. The decryption algorithm calls $P_f(t, c)$. The (obfuscated) program will compute the same PDE key $k = F(K, t)$ as used to create the ciphertext. Then it will use the PDE decryption algorithm and obtain m. This follows via the correctness of the PDE scheme. Finally, it outputs $f(m)$ which is the correct output.

Initial-Encrypt:1

Constants: Puncturable PRF key K.
Input: Randomness $r \in \{0, 1\}^\lambda$.

1. Let $t = \text{PRG}(r)$.
2. Compute: $k = F(K, t)$.
3. Output: (t, k).

Fig. 1. Program Initial-Encrypt:1

Key-Eval:1

Constants: PRF key K, function description $f \in \mathcal{F}$.
Input: (t, c).

1. Compute: $k = F(K, t)$.
2. Output $f(\mathsf{Decrypt}_{\mathsf{PDE}}(k, c))$. (If $\mathsf{Decrypt}_{\mathsf{PDE}}(k, c)$ evaluates to $\perp$ the program outputs $\perp$.)

Fig. 2. Program Key-Eval:1

[5] The program Key-Eval:1 (of Figure 2) is padded to be the same size as Key-Eval:2.

4.1 Proof of Security

Before delving into our formal security proof we give a brief overview with some intuition. In our system a challenge ciphetext CT^* will be a pair (t^*, c^*) of a tag and PDE ciphertext. The first step of our proof is to use pseudorandom generator security to (indetectably) move t^* out of the set of tags $\mathcal{T}$ that might be generated from the program P. (Note the set T corresponds to the possible outputs of the pseudorandom generator.) This then enables us to perform multiple puncturing and hardwiring steps detailed below. Eventually, instead deriving the PDE key k^* as $F(K, t^*)$, it will be chosen uniformly at random. (Here k^* is the PDE key used in creating the challenge ciphertext.)

Furthermore, instead of putting the PDE key k^* into the obfuscated programs given out as keys we will put a punctured version k'. This punctured version is can decrypt all ciphertexts *except* it cannot tell the difference between a PDE encryption of the challenge message m_0 from m_1. However, by the rules of the security game it must be the case that the bit $d_f = f(m_0) = f(m_1)$ for any queried private key function f. Therefore, an obfuscated program for private key f can output d_f when either of the two PDE ciphertexts arises without knowing which one is which. We note that the reduction knows which messages (m_0, m_1) to puncture the PDE key k at since in this security game all keys are given out after the challenge ciphertext is generated.

Finally, at this stage we can simply apply the PDE security game to argue that the message is hidden. We note that the first steps of the proof have similarities to prior programming puncturing proofs [SW14], but we believe the introduction of and the way we utilize puncturable deterministic encryption are novel to this construction. Details of our formal proof are in our full version [Wat14].

5 An Adaptively Secure Construction

We now describe our construction of a functional encryption (FE) scheme that is adaptively secure. We let the message space $\mathcal{M} = \{0,1\}^{\ell(\lambda)}$ for some polynomial function ℓ and the function class be $\mathcal{F}(\lambda) = \mathcal{F}$.

We will use two puncturable PRFs F_1, F_2 such that when we fix the keys K we have that $F_1(K, \cdot)$ takes in a 2λ bit input and outputs two bit strings of length λ and $F_2(K, \cdot)$ takes λ bits to five bitstrings of length λ. In addition, we use a puncturable deterministic encryption scheme where the message space is $\{0,1\}^\lambda$. In our Puncturable PRF and PDE systems master keys are sampled uniformly at random from $\{0,1\}^\lambda$. Finally, we use an indistinguishability secure obfuscator and an *injective* length doubling pseudo random generator $\mathrm{PRG} : \{0,1\}^\lambda \rightarrow \{0,1\}^{2\lambda}$.

Finally, we use a one-bounded secure functional encryption system with master key encryption consisting of algorithms: $\mathsf{KeyGenOB}, \mathsf{EncryptOB}, \mathsf{DecryptOB}$. We assume without loss of generality that the master key is chosen uniformly from $\{0,1\}^\lambda$. The message space $\mathcal{M}$ and key description space $f \in \mathcal{F}$ of the one bounded scheme is the same as the scheme we are constructing.

Our Construction Our construction achieves an adaptively secure functional encryption by bootstrapping from a one-bounded FE scheme that is adaptively secure. At a high level a ciphertext is associated with a tag t and a private key with a tag y. From the pair of tags (t, y) one can (with the proper key material) pseudorandomly derive a master secret key k for a one bounded FE system. The ciphertext will be equipped with an obfuscated program, C, which on input of a key tag y will generate the one bounded key k (associated with the pair (t, y)) and then uses this to create an encryption of the message m under the one-bounded scheme with key k. Likewise, the private key for functionality f comes equipped with an obfuscated program P_f which on input of a ciphertext tag t derives the one bounded secret key k and uses this to create a one-bounded secret key.

The decryption algorithm will pass the key tag y to the ciphertext program to get a one bounded ciphertext $\mathrm{CT}_{\mathrm{OB}}$ and the ciphertext tag t to the key program to get a one bound key $\mathrm{SK}_{\mathrm{OB}}$. Finally, it will apply the one bounded decryption algorithm as $\mathsf{DecryptOB}(\mathrm{CT}_{\mathrm{OB}}, \mathrm{SK}_{\mathrm{OB}})$ to learn the message m. The one bounded key and ciphertext are compatible since they are both derived psuedorandomly from the pair (t, y) to get *same* one-bounded key k. (Note a different pair $(t', y') \neq (t, y)$ corresponds to a different one bounded FE key k' with high probability.)

Our bootstrapping proof structure allows us to develop "selective-ish" techniques at the outer level since in our reductions the ciphertext and private key tags can be chosen randomly ahead of time before the challenge message or any private key queries are known. Then the challenge of dealing with adaptive security is then "pushed down" to the one bounded FE scheme, where it has been solved in previous work [GVW12].

In the description above we have so far omitted one critical ingredient. In addition to generating a one bounded secret key on input t, the program P_f on input t will also generate an encrypted signal a that is passed along with the tag y to the ciphertext program C on decryption to let it know that it is "okay" to generate the one-bounded ciphertext for the pair (t, y). In the actual use of the system, this is the only functionality of the signal. However, looking ahead to our proof we will change the signal encrypted to tell the program C to switch the message for which it generates one bounded encryption encryptions of.

Setup(1^λ)
The algorithm first chooses a random punctured PRF key $K \leftarrow \mathrm{Key}_{F_1}(1^\lambda)$ which is set as the master secret key MSK. Next it creates an obfuscation of the program Initial-Encrypt as $P \leftarrow i\mathcal{O}(1^\lambda, \textsc{Initial-Encrypt}{:}1[K])$.[6]

Encrypt$(\mathrm{PP} = P(\cdot), m \in \mathcal{M})$
The encryption algorithm performs the following steps in sequence.

1. Chooses random $r \in \{0,1\}^\lambda$.

[6] The program $\textsc{Initial-Encrypt}{:}1$ is padded to be the same size as $\textsc{Initial-Encrypt}{:}2$.) This obfuscated program, P serves as the public parameters PP.

2. Sets $(t, K_t, \alpha) \leftarrow P(r)$.
3. Sets $\tilde{\alpha} = \text{PRG}(\alpha)$.
4. Creates the program $C \leftarrow i\mathcal{O}(1^\lambda, \text{CT-}\textsc{Eval}{:}1[K_t, \tilde{\alpha}, m])$.[7]
5. The output ciphertext is $\text{CT} = (t, C)$.

KeyGen$(\text{MSK}, f \in \mathcal{F}(\lambda))$
The KeyGen algorithm first chooses a random $y \in \{0,1\}^\lambda$. It next produces an obfuscated program P_f by obfuscating $P_f \leftarrow i\mathcal{O}(\textsc{Key-Signal}{:}1[K, f, y])$. [8]

The secret key is $\text{SK} = (y, P_f)$.

Decrypt$(\text{CT} = (t, C), \text{SK} = (y, P_f))$
The decryption algorithm takes as input a ciphertext $\text{CT} = (t, C)$ and a secret key $\text{SK} = (y, P_f)$. It first computes $(a, \text{SK}_{\text{OB}}) = P_f(t)$. Next it computes $\text{CT}_{\text{OB}} = C(a, y)$. Finally, it will use the produced secret key to decrypt the produced ciphertext as $\mathsf{DecryptOB}(\text{CT}_{\text{OB}}, \text{SK}_{\text{OB}})$ and outputs the result.

Correctness We briefly sketch a correctness argument. Consider a ciphertext $\text{CT} = (t, C)$ created for message m that is associated with tag t and a key for function f that is associated with tag y. On decryption the algorithm first calls $(a, \text{SK}_{\text{OB}}) = P_f(t)$. Here the obfuscated program computes: $(K_t, \alpha) = F_1(K, t)$, $(d, k, s_1, s_2, s_3) = F_2(K_t, y)$, and $a = \mathsf{Encrypt}_{\mathsf{PDE}}(d, \alpha)$ and $\text{SK}_{\text{OB}} = \mathsf{KeyGenOB}(k, f; s_2)$.

Next, it calls $\text{CT}_{\text{OB}} = C(a, y)$, where C was generated as an obfuscation of program $\text{CT-}\textsc{Eval}{:}1[K_t, \tilde{\alpha}, m]$ where $\tilde{\alpha} = \text{PRG}(\alpha)$. This obfuscated program will compute the same values of $(d, k, s_1, s_2, s_3) = F_2(K_t, y)$ as the key signal program. By correctness of the PDE system we will have that $\mathsf{Decrypt}_{\mathsf{PDE}}(d, a) = \alpha$ and thus the program will output $\mathsf{EncryptOB}(k, m; s_1)$. At this point the decryption algorithm has a one bounded private key for function f and a one bounded ciphertext for message m both created under the same master key k. Therefore, running the one-bounded decryption algorithm will produce $f(m)$.

Initial-Encrypt:1

Constants: Puncturable PRF key K.
Input: Randomness $r \in \{0,1\}^\lambda$.

```
1. Let $t = \text{PRG}(r)$.
2. Compute $(K_t, \alpha) = F_1(K, t)$.
3. Output: $(t, K_t, \alpha)$.
```

Fig. 3. Program Initial-Encrypt:1

[7] The program $\text{CT-}\textsc{Eval}{:}1$ is padded to be the same size as the maximum of $\text{CT-}\textsc{Eval}{:}2$ and $\text{CT-}\textsc{Eval}{:}3$.
[8] The program $\textsc{Key-Signal}{:}1$ is padded to be the same size as $\textsc{Key-Signal}{:}2$.

```
CT-Eval:1

Constants: PRF key $K_t$, $\tilde{\alpha} \in \{0,1\}^{2\lambda}$, message $m \in \mathcal{M}$.
Input: PDE ciphertext $a$ and value $y \in \{0,1\}^{\lambda}$.

1. Compute $(d, k, s_1, s_2, s_3) = F_2(K_t, y)$.
2. Compute $e = \mathsf{Decrypt_{PDE}}(d, a)$.
3. If $\mathrm{PRG}(e) = \tilde{\alpha}$ output $\mathsf{EncryptOB}(k, m; s_1)$.
4. Else output a rejecting $\perp$.
```

Fig. 4. Program CT-Eval:1

```
Key-Signal:1

Constants: PRF key $K$, function description $f \in \mathcal{F}$, tag $y \in \{0,1\}^{\lambda}$.
Input: $t \in \{0,1\}^{2\lambda}$.

1. Compute $(K_t, \alpha) = F_1(K, t)$.
2. Compute $(d, k, s_1, s_2, s_3) = F_2(K_t, y)$.
3. Compute and output $a = \mathsf{Encrypt_{PDE}}(d, \alpha)$ and $\mathrm{SK_{OB}} =$
   $\mathsf{KeyGenOB}(k, f; s_2)$.
```

Fig. 5. Program Key-Signal:1

5.1 Proof of Security

Before delving into our formal security proof we will give a brief intuitive overview of its structure and sequence of games steps. In the first steps of our sequence of games proof we will use pseudorandom generator security to (indetectably) move t^* out of the set of tags $\mathcal{T}$ that might be generated from the program P.[9] Then we use use puncturing techniques to remove the key material, $K^*_{t^*}$, associated with t^* from the obfuscated program given in the public parameters. In addition, the proof will hardwire in the response of all private keys $P_{f_1}, \ldots, P_{f_Q}$ to the input of t^*, where Q is the number of queries issued. These actions are covered in moving from Game 1 to Game 5.

In the next grouping of steps we will introduce a *second alternative message* m_0 into the challenge ciphertext program C^* to go along with the message m_b for $b \in \{0,1\}$. The behavior of the obfuscated program is now (by Game 7) such that if C^* receives an an "α-signal" as input it will output a one-bounded FE encryption of m_b and if it receives a "β-signal" it will output a one-bounded FE encryption of m_0. However, the private key programs P_{f_i} are only set to generate α signals. Before this grouping of steps was executed only α-signals existed.

Subsequently, each private key program P_f is transformed one by one such that they are programmed to send out β-signals upon receiving the tag t^*. When used in decryption this will cause the challenge ciphertext to output one time encryptions of m_0 instead of m_1. Intuitively, this is undetectable because

[9] Note the set T corresponds to the possible outputs of the pseudorandom generator.

$f(m_b) = f(m_0)$ for all private key functions f that can legally be requested. Executing this transformation requires multiple sub steps and is the most complex piece of the proof. It is also where the security one bounded FE scheme is invoked.

Finally, after the above transformations are made we are able to execute two final cleanup steps that remove the message m_b from the ciphertext program C^*. At this point all information about the bit b is removed from the challenge ciphertext and the advantage of any attacker is 0.

Theorem 1. *The above functional encryption scheme is adaptively secure if instantiated with a secure punctured PRF, puncturable deterministic encryption scheme, pseudo random generator, an adaptively secure one-bounded functional encryption scheme and indistinguishability secure obfuscator.*

To prove the above theorem, we first define a sequence of games where the first game is the original FE security game. We begin by with describing Game 1 in detail, which is the adaptive FE security game instantiated with our construction. From there we describe the sequence of games, where each game is described by its modification from the previous game.

In the main body we describe the proof hybrid structure. In our full version [Wat14] we provide the lemmas showing that any poly-time attacker's advantage in each game must be negligibly close to that of the previous game (based on the security of different primitives) .

Game 1 The first game is the original security game instantiated for our construction.

1. Challenger computes keys $K \leftarrow \text{Key}_{F_1}(1^\lambda)$ and randomly chooses the challenge bit $b \in \{0,1\}$.
2. Challenger chooses random $r^* \in \{0,1\}^\lambda$ and computes $t^* = \text{PRG}(r^*)$.
3. Challenger computes $K^*_{t^*}, \alpha^* = F_1(K, t^*)$.
4. Challenger sets $\tilde{\alpha^*} = \text{PRG}(\alpha^*)$.
5. Challenger creates $P \leftarrow i\mathcal{O}(1^\lambda, \textsc{Initial-Encrypt}{:}1[K])$ and passes P to attacker.
6. Phase 1 Queries: Let f_j be the function of associated with the j-th query. Choose random $y_j \in \{0,1\}^\lambda$. Generate the j-th private key by computing $P_{f_j} \leftarrow i\mathcal{O}(\textsc{Key-Signal}{:}1[K, f_j, y_j])$. Output the key as (y_j, P_{f_j}).
7. Attacker gives messages $m_0, m_1 \in \mathcal{M}$ to challenger.
8. Challenger sets the program $C^* \leftarrow i\mathcal{O}(1^\lambda, \text{CT-}\textsc{Eval}{:}1[K^*_{t^*}, \tilde{\alpha^*}, m_b])$.
9. The output ciphertext is $\text{CT} = (t^*, C^*)$.
10. Phase 2 Queries: Same as Phase 1 in step 6.
11. The attacker gives a bit b' and wins if $b' = b$.

Game 2

2. Challenger chooses random $t^* \in \{0,1\}^{2\lambda}$.

Game 3

2. Challenger chooses random $t^* \in \{0,1\}^{2\lambda}$ and sets $K(t^*) = \text{Puncture}_F(K, t^*)$.
5. Challenger creates $P \leftarrow i\mathcal{O}(1^\lambda, \textsc{Initial-Encrypt}{:}2[K(t^*)])$ and passes P to attacker.

Game 4

6. Phase 1 Queries: Let f_j be the function of associated with the j-th query.
   (a) Choose random $y_j \in \{0,1\}^\lambda$.
   (b) Compute $(d_j^*, k_j^*, s_{1,j}^*, s_{2,j}^*, s_{3,j}^*) = F_2(K_{t^*}, y_j)$.
   (c) Compute $a_j^* = \mathsf{Encrypt}_{\mathsf{PDE}}(d_j^*, \alpha^*)$ and $\text{SK}_{\text{OB},j}^* = \mathsf{KeyGenOB}(k_j^*, f_j; s_{2,j}^*)$.
   (d) Compute $P_{f_j} \leftarrow i\mathcal{O}(\textsc{Key-Signal}{:}2[K(t^*), t^*, a_j^*, \text{SK}_{\text{OB},j}^*, f_j, y_j])$.
   (e) Output the key as (y_j, P_{f_j}).
10. Phase 2 Queries: Same as Phase 1 in step 6. (These are also changed as described above.)

Game 5

3. Challenger chooses random $K_{t^*}^*, \alpha^*$.

Game 6

4. Challenger sets $\tilde{\alpha^*} = \text{PRG}(\alpha^*)$ and chooses random $\tilde{\beta^*} \in \{0,1\}^{2\lambda}$.
8. Challenger sets the program $C^* \leftarrow i\mathcal{O}(1^\lambda, \textsc{CT-Eval}{:}2[K_{t^*}^*, \tilde{\alpha^*}, \tilde{\beta^*}, m_b, m_0])$.

Game 7 4. Challenger sets $\tilde{\alpha^*} = \text{PRG}(\alpha^*)$, chooses $\beta^* \in \{0,1\}^\lambda$ at random and sets $\tilde{\beta^*} = \text{PRG}(\beta^*)$.

Game $8, i$ Defined for $i = 0$ to Q. (Q is number of key queries.)

6. Phase 1 Queries: Let f_j be the function of associated with the j-th query.
   (a) Choose random $y_j \in \{0,1\}^\lambda$.
   (b) Compute $(d_j^*, k_j^*, s_{1,j}^*, s_{2,j}^*, s_{3,j}^*) = F_2(K_{t^*}, y_j)$.
   (c) If $j > i$ then set $a_j^* = \mathsf{Encrypt}_{\mathsf{PDE}}(d_j^*, \alpha^*)$; otherwise if $j \leq i$ set $a_j^* = \mathsf{Encrypt}_{\mathsf{PDE}}(d_j^*, \beta^*)$.
   Let $\text{SK}_{\text{OB},j}^* = \mathsf{KeyGenOB}(k_j^*, f_j; s_{2,j}^*)$.
   (d) Compute $P_{f_j} \leftarrow i\mathcal{O}(\textsc{Key-Signal}{:}2[K(t^*), t^*, a_j^*, \text{SK}_{\text{OB},j}^*, f_j, y_j])$.
   (e) Output the key as (y_j, P_{f_j}).

Game 9

4. Challenger chooses $\tilde{\alpha}^* \in \{0,1\}^{2\lambda}$ at random, chooses $\beta^* \in \{0,1\}^\lambda$ at random and sets $\tilde{\beta^*} = \text{PRG}(\beta^*)$.
6. Phase 1 Queries: Let f_j be the function of associated with the j-th query.
   (a) Choose random $y_j \in \{0,1\}^\lambda$.
   (b) Compute $(d_j^*, k_j^*, s_{1,j}^*, s_{2,j}^*, s_{3,j}^*) = F_2(K_{t^*}, y_j)$.
   (c) Set $a_j^* = \mathsf{Encrypt}_{\mathsf{PDE}}(d_j^*, \beta^*)$. Let $\text{SK}_{\text{OB},j}^* = \mathsf{KeyGenOB}(k_j^*, f_j; s_{2,j}^*)$.
   (d) Compute $P_{f_j} \leftarrow i\mathcal{O}(\textsc{Key-Signal}{:}2[K(t^*), t^*, a_j^*, \text{SK}_{\text{OB},j}^*, f_j, y_j])$.
   (e) Output the key as (y_j, P_{f_j}).

Game 10 8. Challenger sets the program $C^* \leftarrow i\mathcal{O}(1^\lambda, \text{CT-Eval:1}[K^*_{t^*}, \tilde{\beta}^*, m_0])$.

We observe at this stage the interaction with the challenger is completely independent of b — note the message m_0 is encrypted regardless of b — and thus the attacker's advantage is 0 in this final game.

Initial-Encrypt:2

Constants: Puncturable PRF key $K(t^*)$.
Input: Randomness r.

```
1. Let $t = \text{PRG}(r)$.
2. Compute $(K_t, \alpha) = F_1(K(t^*), t)$.
3. Output: $(t, K_t, \alpha)$.
```

Fig. 6. Program Initial-Encrypt:2

Key-Signal:2

Constants: PRF key $K(t^*)$, t^*, a^*, SK^*_{OB}, function description f, tag $y \in \{0,1\}^\lambda$.
Input: $t \in \{0,1\}^{2\lambda}$.

```
1. If $t = t^*$ output $a^*, \text{SK}^*_{\text{OB}}$.
2. Compute $(K_t, \alpha) = F_1(K(t^*), t)$.
3. Compute $(d, k, s_1, s_2, s_3) = F_2(K_t, y)$.
4. Compute and output $a = \mathsf{Encrypt}_{\mathsf{PDE}}(d, \alpha)$ and $\text{SK}_{\text{OB}} = \mathsf{KeyGenOB}(k, f; s_2)$
```

Fig. 7. Program Key-Signal:2

CT-Eval:2

Constants: PRF key K_t, $\tilde{\alpha}, \tilde{\beta} \in \{0,1\}^{2\cdot\lambda}$, messages $m, m_{\text{fixed}} \in \mathcal{M}$.
Input: (a, y).

```
1. Compute $(d, k, s_1, s_2, s_3) = F_2(K_t, y)$.
2. Compute $e = \mathsf{Decrypt}_{\mathsf{PDE}}(d, a)$.
3. If $\text{PRG}(e) = \tilde{\alpha}$ output $\mathsf{EncryptOB}(k, m; s_1)$.
4. If $\text{PRG}(e) = \tilde{\beta}$ output $\mathsf{EncryptOB}(k, m_{\text{fixed}}; s_3)$.
5. Else output a rejecting $\perp$.
```

Fig. 8. Program CT-Eval:2

References

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. `http://eprint.iacr.org/`.

[ABSV14] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. The trojan method in functional encryption: From selective to adaptive security, generically. Cryptology ePrint Archive, Report 2014/917, 2014. `http://eprint.iacr.org/`.

[BCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, pages 52–73, 2014.

[BFO08] Alexandra Boldyreva, Serge Fehr, and Adam O'Neill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In *CRYPTO*, pages 335–359, 2008.

[BFOR08] Mihir Bellare, Marc Fischlin, Adam O'Neill, and Thomas Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *CRYPTO*, pages 360–378, 2008.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013.

[BS14] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. Cryptology ePrint Archive, Report 2014/550, 2014. `http://eprint.iacr.org/`.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: definitions and challenges. In *Proceedings of the 8th conference on Theory of cryptography*, TCC'11, pages 253–273, 2011.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 535–554, Berlin, Heidelberg, 2007. Springer-Verlag.

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

[BWZ14] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930, 2014. `http://eprint.iacr.org/`.

[CHL$^+$14] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehle. Cryptanalysis of the multilinear map over the integers. Cryptology ePrint Archive, Report 2014/906, 2014. `http://eprint.iacr.org/`.

[CLT14] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. Cryptology ePrint Archive, Report 2014/975, 2014. `http://eprint.iacr.org/`.

[CW14] Jie Chen and Hoeteck Wee. Semi-adaptive attribute-based encryption and improved delegation for boolean formula. SCN (To appear), 2014. `http://eprint.iacr.org/`.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *CRYPTO*, pages 518–535, 2014.

[GGHZ14a] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure attribute based encryption from multilinear maps. Cryptology ePrint Archive, Report 2014/622, 2014. `http://eprint.iacr.org/`.

[GGHZ14b] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666, 2014. `http://eprint.iacr.org/`.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.

[GHMS14] Craig Gentry, Shai Halevi, Hemanta K. Maji, and Amit Sahai. Zeroizing without zeroes: Cryptanalyzing multilinear maps without encodings of zero. Cryptology ePrint Archive, Report 2014/929, 2014. `http://eprint.iacr.org/`.

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

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

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, EUROCRYPT'08, 2008.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

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

[SW08] Amit Sahai and Brent Waters. Slides on functional encryption. PowerPoint presentation, 2008. `http://www.cs.utexas.edu/~bwaters/presentations/files/functional.ppt`.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[Wat14] Brent Waters. A punctured programming approach to adaptively secure functional encryption. Cryptology ePrint Archive, Report 2014/588, 2014. `http://eprint.iacr.org/`.