Functional Encryption: Definitions and Challenges

Dan Boneh* Stanford University
Amit Sahai UCLA
Brent Waters † University of Texas at Austin

Abstract

We initiate the formal study of functional encryption by giving precise definitions of the concept and its security. Roughly speaking, functional encryption supports restricted secret keys that enable a key holder to learn a specific function of encrypted data, but learn nothing else about the data. For example, given an encrypted program the secret key may enable the key holder to learn the output of the program on a specific input without learning anything else about the program.

We show that defining security for functional encryption is non-trivial. First, we show that a natural game-based definition is inadequate for some functionalities. We then present a natural simulation-based definition and show that it (provably) cannot be satisfied in the standard model, but can be satisfied in the random oracle model. We show how to map many existing concepts to our formalization of functional encryption and conclude with several interesting open problems in this young area.

*Supported by NSF, MURI, and the Packard foundation.

†Supported by NSF CNS-0716199, CNS-0915361, and CNS-0952692, Air Force Office of Scientific Research (AFO SR) under the MURI award for "Collaborative policies and assured information sharing" (Project PRESIDIO), Department of Homeland Security Grant 2006-CS-001-000001-02 (subaward 641), and the Alfred P. Sloan Foundation.

1 Introduction

Encryption is a method for a user to securely share data over an insecure network or storage site. Before the advent of public key cryptography, a widely held view was that for two users to communicate data confidentially they would need to a priori establish a mutually held secret key k. While this might be acceptable for some small or tightly knit organizations, such a solution was clearly infeasible for larger networks such as today's Internet consisting of billions of users. Over thirty years ago, Diffie and Hellman [DH76a, DH76b] put forth a radically new idea in the concept of public key cryptography, where two parties can securely communicate with each other *without* having an a prior mutual secret — radically challenging the conventional wisdom of the time.

Today public key encryption is an invaluable tool and its use is ubiquitous in building tools from secure web communication (e.g., SSH, SSL), to disk encryption, and secure software patch distribution. However, there is an ingrained view that: (1) *Encryption is a method to send a message or data to a single entity holding a secret key*, and (2) *Access to the encrypted data is all or nothing – one can either decrypt and read the entire plaintext or one learns nothing at all about the plaintext other than its length.*

For many emerging applications such as "cloud" services this notion of public-key encryption is insufficient. For example, there is often a need to specify a decryption policy in the ciphertext and only individuals who satisfy the policy can decrypt. More generally, we may want to only give access to a function of the plaintext, depending on the decryptor's authorization. As a concrete example, consider a cloud service storing encrypted images. Law enforcement may require the cloud to search for images containing a particular face. Thus, the cloud needs a restricted secret key that decrypts images that contain the target face, but reveals nothing about other images. More generally, the secret key may only reveal a function of the plaintext image, for example an image that is blurred everywhere except for the target face. Traditional public-key cryptography cannot help with such tasks.

We believe that it is time to adopt a new broad vision of encryption systems. To this end, we explore the concept of *functional encryption*. In a functional encryption system, a decryption key allows a user to learn a *function* of the encrypted data. Briefly, in a functional encryption system for functionality $F(\cdot,\cdot)$ (modeled as a Turing Machine) an authority holding a master secret key can generate a key sk_k that enables the computation of the function $F(k,\cdot)$ on encrypted data. More precisely, using sk_k the decryptor can compute $F(k,x)$ from an encryption of x. Intuitively, the security of the system guarantees that one cannot learn anything more about x, but as we shall see, capturing this rigorously is quite challenging.

We can now see the power of functional encryption. Let us consider what can be achieved if we could realize functional encryption for any polynomial-time Turing Machine $F(\cdot,\cdot)$. In applications of access control, one could let $x = (\mathsf{ind}, m)$ encode a message m as well as an arbitrarily complex access control program ind that will act over the description of a user's credentials. The functionality F would interpret the program ind over k and output the message m if and only if ind accepts on input k. Moreover, the program ind would be hidden and thus one would not necessarily know why decryption was successful or what other keys would satisfy ind. We give many more examples in Section 3.

Our Contributions Recently, there have been multiple systems that suggest moving beyond the traditional boundaries of encryption. Some examples include Identity-Based Encryption [Sha84, BF03, Coc01], searchable encryption [BCOP04] and Attribute-Based Encryption [SW05]. These and other related works such as [BW07, KSW08] propose specific new systems for problems ranging from expressive access con-

trol to searching on encrypted data. In the last few years, the term "functional encryption[1]" was adopted to describe this new area [LOS+10, OT10, AL10].

While these results contain special cases of functional encryption, the general concept has never been formally defined or studied. In this paper we put forth a formal treatment of the subject and discuss many of the remaining challenges. We begin with a general framework and syntax for functional encryption and show how existing encryption concepts, such as attribute based encryption and many others, can be elegantly expressed as particular functionalities of functional encryption.

Defining security of abstract functional encryption turns out to be highly non-trivial. We begin with a natural indistinguishability game-based definition (based on a definition of secure predicate encryption from [BW07, KSW08]). Unfortunately, we show that this simple definition is inadequate for certain functionalities since trivially insecure constructions may satisfy it.

Given the inadequacy of game-based definitions we move to simulation-based definitions in the spirit of the original notion of semantic security of Goldwasser and Micali [GM84]. The goal is to capture the notion that the adversary learns nothing about the plaintext other than functions $F(k, \cdot)$ of the plaintext for which he has a secret key. Somewhat surprisingly, we show a connection to non-committing encryption [CFGN96, Nie02] which proves that our definition cannot be satisfied for the same reason that non-interactive non-committing encryption is impossible. However, we show that our definition can be satisfied in the random oracle model, and we exhibit constructions for interesting functionalities that can be shown to be secure. (Independently, O'Neill [O'N10] also observed a gap between simulation and game-based definitions and a connection to non-committing encryption.)

Functional encryption is still in its infancy and many fascinating open problems remain. We conclude with several directions for future work. The key challenge is the construction of functional encryption for more general functionalities. Another important question is understanding the relative power of functionalities: when does one functionality imply another and when can functionalities be black-box separated?

2 Functional Encryption Syntax

We begin by describing the syntactic definition of functional encryption (FE) for a functionality F. The functionality F describes the functions of a plaintext that can be learned from the ciphertext. More precisely, a functionality is defined as follows.

Definition 1. *A functionality F defined over (K, X) is a function $F : K \times X \to \{0,1\}^*$ described as a (deterministic) Turing Machine. The set K is called the* key space *and the set X is called the* plaintext space*. We require that the key space K contain a special key called the* empty key *denoted ϵ.*

A functional encryption scheme for the functionality F enables one to evaluate $F(k, x)$ given the encryption of x and a secret key sk_k for k. The algorithm for evaluation $F(k, x)$ using sk_k is called *decrypt*. More precisely, a functional encryption scheme is defined as follows.

Definition 2. *A functional encryption scheme (FE) for a functionality F defined over (K, X) is a tuple of four PPT algorithms* $(\mathsf{setup}, \mathsf{keygen}, \mathsf{enc}, \mathsf{dec})$ *satisfying the following correctness condition for all $k \in K$ and $x \in X$:*

[1] We note that both the term "functional encryption" and its underlying concept were introduced by the authors of this paper. This term was first publicly used to describe the line of work starting with [SW05] in a talk "Functional Encryption: Beyond Public Key Cryptography" [Wat08] in 2008, given by one of the authors of this paper.

$(pp, mk) \leftarrow \text{setup}(1^\lambda)$ *(generate a public and master secret key pair)*

$sk \leftarrow \text{keygen}(mk, k)$ *(generate secret key for k)*

$c \leftarrow \text{enc}(pp, x)$ *(encrypt message x)*

$y \leftarrow \text{dec}(sk, c)$ *(use sk to compute $F(k, x)$ from c)*

then we require that $y = F(k, x)$ with probability 1.

We define security of a functional encryption scheme in Section 4. For now, we briefly show that standard public-key encryption is a simple example of functional encryption. Let $K := \{1, \epsilon\}$ and consider the following functionality F defined over (K, X) for some plaintext space X:

$$F(k, x) := \begin{cases} x & \text{if } k = 1 \\ \text{len(x)} & \text{if } k = \epsilon \end{cases}$$

A secret key for $k = 1$ fully decrypts valid ciphertexts, while the empty key $k = \epsilon$ simply returns the bit length of the plaintext. Hence, this functionality syntactically defines standard public-key encryption.

The empty key ϵ: The special key ϵ in K captures all the information about the plaintext that intentionally leaks from the ciphertext, such as the length of the encrypted plaintext. The secret key for ϵ is empty and also denoted by ϵ. Thus, anyone can run $\text{dec}(\epsilon, c)$ on a ciphertext $c \xleftarrow{R} \text{enc}(\text{pp}, x)$ and obtain all the information about x that intentionally leaks from c.

Further parametrization. In some cases the key space K and plaintext space X are further parametrized by quantities generated by the setup algorithm. For example, setup may output an RSA modulus N in which case the sets K and X and the functionality F are defined as tuples over $\mathbb{Z}_N$. More generally, we allow setup to output a third parameter π and we denote the key and plaintext space by K_π and X_π. The functionality F is the defined as

$$F_\pi : K_\pi \times X_\pi \to \{0,1\}^* .$$

When π is clear from context, we avoid writing it as an explicit subscript.

2.1 Sub-classes of functional encryption

So far we defined the most general syntax for a functional encryption scheme. For the applications we have in mind it is convenient to define two sub-classes of functional encryption where the plaintext space X has additional structure.

Predicate encryption [BW07, KSW08]. In many applications a plaintext $x \in X$ is itself a pair $(\mathsf{ind}, m) \in I \times M$ where ind is called an index and m is called the payload message. For example, in an email system the index might be the sender's name while the payload is the email contents.

In this context, an FE scheme is defined in terms of a polynomial-time predicate $P : K \times I \to \{0, 1\}$ where K is the key space. More precisely, the FE functionality over $(\, K \cup \{\epsilon\},\ \ (I \times M)\,)$ is defined as

$$F\big(k \in K,\ \ (\mathsf{ind}, m) \in X\ \big) := \begin{cases} m & \text{if } P(k, \mathsf{ind}) = 1, \text{ and} \\ \perp & \text{if } P(k, \mathsf{ind}) = 0 \end{cases}$$

Consequently, let c be an encryption of (ind, m) and let sk_k be a secret key for $k \in K$. Then $\text{dec}(\text{sk}_k, c)$ reveals the payload in c when $P(k, \mathsf{ind}) = 1$ and reveals nothing new about m otherwise.

Predicate encryption with public index. A sub-class of predicate encryption makes the plaintext index easily readable from the ciphertext. In particular, in this type of FE the empty key ϵ explicitly reveals the index ind, namely

$$F\big(\epsilon,\ (\mathsf{ind}, m)\ \big) = (\mathsf{ind},\ \mathsf{len}(m)\ )$$

Hence, $\mathrm{dec}(\epsilon, c)$ gives anyone the index component of the plaintext as well as the bit length of m.

3 Capturing Cryptosystems in the Context of Functional Encryption

Many recent encryption concepts and constructions can be viewed as special cases of Functional Encryption. In this section we give a few examples to show how functional encryption captures these encryption concepts. Security of these schemes is captured by the general security definitions of functional encryption in the next section.

3.1 Predicate encryption systems with public index

The first class of systems that we consider are Predicate encryption schemes with public index.[2] We begin our study with the simplest interesting case of Identity-Based Encryption and then advance to more expressive methods of access formulas. We will describe these systems using the notation for predicate encryption defined in Subsection 2.1. [3]

Identity-Based Encryption. In Identity-Based Encryption (IBE) [Sha84] ciphertexts and private keys are associated with strings (a.k.a identities) and a key can decrypt a ciphertext if the two strings are equal. IBE represents the first functionality that is not directly realizable from public key encryption [BPR+08]. IBE is formally described as a Predicate Encryption scheme where:

1. The key space K is $K := \{0,1\}^* \cup \{\epsilon\}$.
2. The plaintext is a pair (ind, m) where the index space $I := \{0,1\}^*$.
3. The predicate P on $K \times I$ is defined as $\quad P\big(k \in K \setminus \{\epsilon\},\ \mathsf{ind} \in I\ \big) := \begin{cases} 1 & \text{if } k = \mathsf{ind}, \text{ and} \\ 0 & \text{otherwise} \end{cases}$

Boneh and Franklin [BF03] and Cocks [Coc01] construct the first practical IBE systems, which were proven secure according to an indistinguishability definition that is a special case of our definition of functional encryption security (Definition 3 in Section 4). These first schemes were proven secure in the random oracle model. Subsequent schemes were proven secure in the standard model, but under a weaker notion known as selective security [CHK03, BB04a], and further subsequent systems were proven adaptively secure [BB04b, Wat05, Gen06]. Recently, there have been multiple lattice-based constructions of IBE systems [GPV08, CHKP10, ABB10].

For these systems to properly support the empty key ϵ function, the ciphertext must explicitly include ind and the length of the message in the clear.

[2] This class has also been informally referred to as "payload hiding" [BW07, KSW08] in the literature.

[3] Recall that for all predicate encryption schemes with public index we have that $F\big(\epsilon,\ (\mathsf{ind}, m)\ \big) = (\mathsf{ind},\ \mathsf{len}(m)\ )$.

Attribute-Based Encryption. Sahai and Waters [SW05] proposed a notion of encryption, called Attribute-Based Encryption (ABE), where one could express complex access policies. Subsequently, Goyal, Pandey, Sahai and Waters [GPSW06] refined this concept into two different formulations of ABE: Key Policy ABE and Ciphertext-Policy ABE.

We first describe Key-Policy ABE for boolean formulas, as was realized by Goyal et. al. [GPSW06]. [4] A Key-Policy ABE system over n variables can be described as a predicate encryption scheme (with public index) for the predicate $P_n : K \times I \to \{0,1\}$ where:

1. The key space K is the set of all poly-sized boolean formulas ϕ in n variables $\vec{z} = (z_1, \ldots, z_n) \in \{0,1\}^n$. We let $\phi(\vec{z})$ denote the value of the formula ϕ at $\vec{z}$.

2. The plaintext is a pair $(\mathsf{ind} = \vec{z},\ m)$ where the index space is $I := \{0,1\}^n$, and where we interpret $\vec{z}$ as a bit vector representing the boolean values $z_1, \ldots z_n$.

3. The predicate P_n on $K \times I$ is defined as

$$P_n\big(\phi \in K \setminus \{\epsilon\},\ \ \mathsf{ind} = \vec{z} \in I\ \big) := \begin{cases} 1 & \text{if } \phi(\vec{z}) = 1, \text{ and} \\ 0 & \text{otherwise} \end{cases}$$

In these systems the key provides an access formula that operates over a set of n attributes that must evaluate to true for decryption to yield the message m. Goyal et al. also describe how to construct a "Large Universe" construction where ind can be viewed as a set of strings. Then K consists of all monotone boolean formulas over strings. To evaluate $\phi(\mathsf{ind})$ we evaluate a leaf labeled with string x in ϕ as '0' if $x \notin \mathsf{ind}$.

Ciphertext-Policy ABE. A dual concept of Attribute-Based Encryption is Ciphertext-Policy Attribute-Based Encryption (CP-ABE), where the roles of the ciphertext and key are essentially reversed. A Ciphertext-Policy ABE system over n variables can be described as predicate encryption scheme (with public index) for the predicate $P_n : K \times I \to \{0,1\}$ where:

1. The key space $K := \{0,1\}^n$ is the set of all n bit strings representing n boolean variables $\vec{z} = (z_1, \ldots, z_n) \in \{0,1\}^n$.

2. The plaintext is a pair $(\mathsf{ind} = \phi,\ m)$ where the index space I is the set of all poly-sized boolean formulas ϕ over n variables.

3. The predicate P_n on $K \times I$ is defined as

$$P_n\big(\vec{z} \in K \setminus \{\epsilon\},\ \ \mathsf{ind} = \phi \in I\ \big) := \begin{cases} 1 & \text{if } \phi(\vec{z}) = 1, \text{ and} \\ 0 & \text{otherwise} \end{cases}$$

CP-ABE systems are constructed in [BSW07, GJPS08, Wat11]. Most constructions of ABE (both Ciphertext-Policy and Key-Policy) were proven secure in a weaker selective model of security. Recently Lewko et. al. [LOS+10] showed how to give fully secure realizations meeting our security definition.

[4] The ABE solutions of Goyal et. al. and others [BSW07, OSW07, GJPS08, Wat11] actually extend to formulas over threshold gates and to Monotone Span Programs; however, we restrict our description to Boolean formulas for simplicity.

3.2 Predicate Encryption Systems

While the previous systems described allow for expressive forms of access control, they are limited in two ways. First, the policy ind is given in the clear as part of the empty functionality — often this in itself can be considered sensitive. Second, it does not allow for computation on the encrypted data, which might include such applications as search. Here we describe current Predicate Encryption systems that do not leak the index ind.

Anonymous Identity-Based Encryption. The problem of Anonymous Identity-Based Encryption was first proposed by Boneh et. al. [BCOP04] and later formalized by Abdalla et. al. [ABC$^+$08]. Other constructions include [BW06, Gen06, CHKP10, ABB10]. The functionality of Anonymous IBE is similar to IBE except that the string representing the ciphertext identity is hidden and one can only determine it if they have the corresponding private key. Therefore, we can describe Anonymous IBE in the exact same manner as above, except we have that $F\big(\epsilon,\ (\mathsf{ind}, m)\ \big) = \text{len}(m)$. The empty functionality only gives the message length, but ind stays hidden.

Hidden Vector Encryption. Boneh and Waters [BW07] proposed what they called a hidden vector encryption system. In such a system a ciphertext contains a vector of n elements in $\{0,1\}^*$ and a private key contains of a vector of n elements in $\{*\} \cup \{0,1\}^*$ where we refer to $*$ as a wildcard character. More precisely,

1. The key space K is all $(v_1, \ldots v_n)$ where each $v_i \in \{*\} \cup \{0,1\}^*$.
2. The plaintext is a pair $(\mathsf{ind} = (w_1, \ldots, w_n),\ m)$ where each $w_i \in \{0,1\}^*$. The index space in $I := (\{0,1\}^*)^n$.
3. The predicate P_n on $K \times I$ is defined as

$$P_n\big(\ (v_1, \ldots, v_n) \in K \setminus \{\epsilon\},\ \ \mathsf{ind} = (w_1, \ldots, w_n)\ \big) := \begin{cases} 1 & \text{if } v_i = w_i \text{ whenever } v_i \neq *, \text{ and} \\ 0 & \text{otherwise} \end{cases}$$

Applications of the predicate include conjunctive and range searches. Independently, Shi et. al. [SBC$^+$07] proposed a related system in a weaker security model. Again we note that $F\big(\epsilon,\ (\mathsf{ind}, m)\ \big) = \text{len}(m)$ so that ciphertexts do not reveal ind.

Inner Product Predicate. The previous system was limited to conjunctive searches. Katz, Sahai and Waters [KSW08] proposed a system for testing if a dot product operation over the ring Z_N is equal to 0, where N is the product of three random primes chosen by the setup algorithm. This enabled more complex evaluations on disjunctions, polynomials, and CNF/DNF formulae. Subsequently, Okamoto and Takashima [OT09] and Lewko et. al. [LOS$^+$10] gave constructions over the field F_p. We describe this predicate for vectors of length n.

1. The setup algorithm defines a randomly chosen prime p of length κ, where κ is the security parameter.
2. The key space K is all $\vec{v} = (v_1, \ldots v_n)$ where each $v_i \in F_p$.
3. The plaintext is a pair $(\mathsf{ind} = (w_1, \ldots, w_n),\ m)$ where each $w_i \in F_p$. The index space is $I := (F_p)^n$.

4. The predicate $P_{n,p}$ on $K \times I$ is defined as

$$P_{n,p}\big(\ (v_1,\ldots,v_n) \in K \setminus \{\epsilon\},\ \ \mathsf{ind} = (w_1,\ldots,w_n)\ \big) := \begin{cases} 1 & \text{if } \sum_{i=1,\ldots,n} v_i \cdot w_i = 0 \\ 0 & \text{otherwise} \end{cases}$$

3.3 Other systems and combinations

Different researchers have realized different combinations of the above core systems. Examples of these include combinations of: Attribute-Based Encryption and Broadcast Encryption [AI09], Identity-Based Broadcast Encryption [Del07, DPP07, SF07, GW09], broadcast HIBE [BH08], and Inner-Product Encryption and ABE [OT10]. All are captured as special cases of functional encryption.

4 Security definitions

Given the syntactic definitions of Functional Encryption (FE) from Section 2 we now turn to defining security of an FE scheme. In this section we give game based definitions. In Section 5 we discuss simulation-based definitions.

Let $\mathcal{E}$ be an FE scheme for functionality F defined over (K, X). Our goal is to define security against an adaptive adversary that repeatedly asks for secret keys sk_k for keys $k \in K$ of the attacker's choice. As we shall see, defining security against such attackers is more delicate than one might first expect. The problem is how to define the challenge ciphertext in a semantic security game. As usual, once the attacker obtains all the secret keys he desires, he outputs two challenge messages $m_0, m_1 \in X$ and expects to get back an encryption c of m_0 or m_1 chosen at random by the challenger. Clearly, if the attacker has a secret key sk_k for some $k \in K$ for which $F(k, m_0) \neq F(k, m_1)$ then he can easily answer the challenge c by outputting

$$\begin{cases} 0 & \text{if } \text{dec}(\text{sk}_k, c) = F(k, m_0), \text{ and} \\ 1 & \text{otherwise} \end{cases}$$

Hence, for the definition to be satisfiable we must severely restrict the attacker's choice of m_0, m_1 and require that they satisfy

$$F(k, m_0) = F(k, m_1) \quad \text{for all } k \text{ for which the attacker has } \text{sk}_k. \tag{1}$$

Since the empty key ϵ reveals the plaintext length, condition (1) ensures that $|m_0| = |m_1|$, as in the standard PKE definition of semantic security.

Security definition. With requirement (1) in place we obtain a natural game for defining security of an FE scheme $\mathcal{E}$. For $b = 0, 1$ define experiment b for an adversary $\mathcal{A}$ as follows:

- Setup: run $(\text{pp}, \text{mk}) \leftarrow \text{setup}(1^\lambda)$ and give pp to $\mathcal{A}$.
- Query: $\mathcal{A}$ adaptively submits queries k_i in K for $i = 1, 2, \ldots$ and is given $\text{sk}_i \leftarrow \text{keygen}(\text{mk}, k_i)$.
- Challenge: $\mathcal{A}$ submits two messages $m_0, m_1 \in X$ satisfying (1) and is given $\text{enc}(\text{pp}, m_b)$.
- $\mathcal{A}$ continues to issue key queries as before subject to (1) and eventually outputs a bit in $\{0, 1\}$.

For $b = 0, 1$ let W_b be the event that the adversary outputs 1 in Experiment b and define

$$\mathrm{FE}_{\mathrm{adv}}[\mathcal{E}, \mathcal{A}](\lambda) := \big|\Pr[W_0] - \Pr[W_1]\big|$$

Definition 3. *An FE scheme $\mathcal{E}$ is secure if for all PPT $\mathcal{A}$ the function $\mathrm{FE}_{adv}[\mathcal{E}, \mathcal{A}](\lambda)$ is negligible.*

Definition 3 is a generalization of related definitions from [BW07, KSW08].

4.1 A "brute force" construction

We briefly show that any functionality F where the key space K has polynomial size can be easily realized. Write $s = |K| - 1$ and $K = \{\epsilon, k_1, \ldots, k_s\}$. In this brute force construction, the size of public parameters, secret key, and ciphertext are all proportional to s. A closely related construction is given in [BW07].

The brute force FE scheme realizing F uses a semantically secure public-key encryption scheme (G, E, D) and works as follows:

- setup(1^λ): for $i = 1, \ldots, s$ run $(\mathrm{pp}_i, \mathrm{mk}_i) \leftarrow G(1^\lambda)$.

  $$\text{output:} \qquad \mathrm{pp} := (\mathrm{pp}_1, \ldots, \mathrm{pp}_s) \qquad \text{and} \qquad \mathrm{mk} := (\mathrm{mk}_1, \ldots, \mathrm{mk}_s)$$

- keygen(mk, k_i): output $\mathrm{sk}_i := \mathrm{mk}_i$.
- enc(pp, x): output $\vec{c} := \big(\, F(\epsilon, x),\ E\big(\mathrm{pp}_1, F(k_i, x)\big),\ \ldots,\ E\big(\mathrm{pp}_s, F(k_s, x)\big)\,\big)$.
- dec$(\mathrm{sk}_i, \vec{c})$: output c_0 if $\mathrm{sk}_i = \epsilon$, and output $D(\mathrm{sk}_i, c_i)$ otherwise.

Clearly, a ciphertext $\vec{c}$ leaks the bit lengths of $F(k_i, x)$ for $i = 1, \ldots, s$. Therefore, for this construction to be secure we must assume that this information is already leaked by the empty functionality $F(\epsilon, \cdot)$, namely that $|F(k_i, x)|$ for $i = 1, \ldots, s$ is contained in $F(\epsilon, x)$. If so then we say that F reveals functional bit lengths.

Theorem 1. *Let F be a functionality that reveals functional bit lengths. If (G, E, D) is a semantically secure public-key encryption scheme then the brute force FE system implementing F is secure.*

Proof Sketch. The proof is by a standard hybrid argument across the s components of the challenge ciphertext. □

4.2 Insufficiency of the game-based security definition

We will now show that for certain complex functionalities Definition 3 is too weak. For these functionalities we construct systems that are secure under Definition 3, but should not be considered secure. Nevertheless, for functionalities such as predicate encryption *with public index* we show in Section 5 that Definition 3 is adequate.

We give a simple example of a functionality for which the game-based Definition 3 is insufficient. Let π be a one-way permutation and consider the functionality F that only admits the trivial key ϵ, defined as follows:

$$F(\epsilon, x) = \pi(x)$$

It is clear that the "right" way to achieve functional encryption for this very simple functionality is to have the functional encryption algorithm itself simply output $\pi(x)$ on input x, namely $\mathsf{enc}(\mathsf{pp}, x) = \pi(x)$. This scheme would also clearly achieve the simulation-based definition of security presented in Section 5.

However, consider an "incorrect" realization of this functionality where the functional encryption algorithm outputs x on input x, namely $\mathsf{enc}(\mathsf{pp}, x) = x$. Clearly this system leaks more information about the plaintext than needed. Nevertheless, it is easy to verify that this construction satisfies the game-based definition from Section 4. This is because for any two values x and y, it is the case that $F(\epsilon, x) = F(\epsilon, y)$ if and only if $x = y$ and therefore the attacker can only issue challenge messages m_0, m_1 where $m_0 = m_1$.

This problematic system, however, would clearly not achieve the simulation-based definition of security presented in Section 5, since if x is chosen at random, the real-life adversary would be able to recover x always, while the simulator would not be able to recover x without breaking the one-wayness of the permutation π.

While the simple example above may seem to be "abusing" the role of the trivial key ϵ, it is easy to modify the functionality example F above so that there is exactly one non-trivial key $k \in K$ that outputs $\pi(x)$. The only difference to the construction above would be that the functional encryption algorithm would output a public-key encryption[5] of either $\pi(x)$ (in the "correct" implementation) or x (in the "incorrect" implementation), and the secret key for key k would be the secret key of the public-key encryption scheme. Again, it is easy to verify that the incorrect implementation satisfies the game-based definition.

Discussion. What does this separation show? While this is a subjective question, our view is that it shows that if the output of the functionality is supposed to have some computational hiding properties – that is, security of your application is not only based on the information-theoretic properties of the function, but also on the computational properties of the function – then there is a real problem with the game-based formulation of security. The game-based formulation essentially ignores any computational hiding properties of the function, and therefore offers no security guarantees that could be meaningfully combined with such computational considerations.

5 Simulation based definitons

In this section, we explore security definitions for functional encryption that arise from the simulation paradigm [GM84, GMR85, GMW86] that has served us so well, especially in the closely related context of secure computation protocols.

We begin by considering a simulation-based definition[6] of security for functional encryption that captures the most basic intuition we have: That getting the secret key sk_k corresponding to the key $k \in K$ should only reveal $F(k, x)$ when given an encryption of x.

It turns out that we can achieve this simulation-based definition for natural functionalities in the *random oracle model*, where in the ideal model the random oracle would also be simulated. We argue that in fact this (very strong) random oracle model seems necessary for a meaningful simulation-based definition of security for functional encryption: we show that even in the *non-programmable* random oracle model (where the simulator, too, only has oracle access to the same random oracle that is provided to the distinguisher), simulation-secure functional encryption (for a seemingly "minimal" formulation of simulation-security)

[5] The public-key encryption would need to be non-committing to achieve the simulation-based definition of security for the good case.

[6] We note that there are several natural variants possible for such a definition. We have chosen a definition that is strong in the sense that it requires a universal black-box simulator. We will later discuss some weaker formulations.

even just for the IBE functionality is impossible to achieve. At a high level, this is because any simulation-based definition that allows the adversary to query for secret keys after seeing the challenge ciphertext must achieve something very similar in spirit to *non-interactive non-committing encryption*, where exactly these kinds of impossibility and possibility results are known [Nie02].

In our main definition below (that we will achieve in our positive results), we will use some non-standard syntax for representing a *stateful oracle*[7]. When we write $A^{B(\cdot)[[x]]}$, we mean that the algorithm A can issue a query q to its oracle, at which point $B(q, x)$ will be executed and output a pair (y, x'). The value y is then communicated to A as the response to its query, and the variable x is set to x', and this updated value is fed to the algorithm B the next time it is queried as an oracle, and fed to any algorithms executed later in an experiment that want x as an input. Also, if we write $A^{B^\circ(\cdot)}$, we mean that A can send a query q to its oracle, at which point $B^\circ(q)$ is executed, and any oracle queries that B makes are answered by A.

Definition 4. *An FE scheme $\mathcal{E}$ is simulation-secure if there exists an (oracle) PPT algorithm $Sim = (Sim_1, Sim_O, Sim_2)$ such that for any (oracle) PPT algorithms $Message$ and Adv, we have that the following two distribution ensembles (over the security parameter λ) are computationally indistinguishable:*

Real Distribution:

1. $(pp, mk) \leftarrow \text{setup}(1^\lambda)$
2. $(\vec{x}, \tau) \leftarrow Message^{\text{keygen}(mk,\cdot)}(pp)$
3. $\vec{c} \leftarrow \text{enc}(pp, \vec{x})$
4. $\alpha \leftarrow Adv^{\text{keygen}(mk,\cdot)}(pp, \vec{c}, \tau)$
5. *Let $y_1, \ldots, y_\ell$ be the queries to* keygen *made by $Message$ and Adv in the previous steps.*
6. *Output* $(pp, \vec{x}, \tau, \alpha, y_1, \ldots, y_\ell)$

Ideal Distribution:

1. $(pp, \sigma) \leftarrow Sim_1(1^\lambda)$
2. $(\vec{x}, \tau) \leftarrow Message^{SimO(\cdot)[[\sigma]]}(pp)$
3. $\alpha \leftarrow Sim_2^{F(\cdot,\vec{x}),\ Adv^\circ(pp,\cdot,\tau)}(\sigma, F(\epsilon, \vec{x}))$
4. *Let $y_1, \ldots, y_\ell$ be the queries to F made by Sim in the previous steps*[8].
5. *Output* $(pp, \vec{x}, \tau, \alpha, y_1, \ldots, y_\ell)$

We note that this definition can be extended further to allow the adversary to receive challenge ciphertexts adaptively (instead of as a single vector), and all our positive results below would extend to this setting. We omit this generalization due to the notational complexity that would be required to formulate such a definition.

[7] The more standard way to formalize this communication structure would be through interactive Turing Machines, but we find this notation to be simpler to parse.

[8] Note that Sim does not need to query the oracle for $F(\epsilon, \vec{x})$, as this is provided as an explicit input to Sim_2. We choose this formulation since in the real distribution, the Adversary does not explicitly need to ask keygen for the key corresponding to ϵ in order to gain knowledge about $F(\epsilon, \vec{x})$.

5.1 Impossibility of simulation-secure functional encryption

In this section, we briefly sketch the impossibility result for simulation-secure functional encryption, even for a quite simple functionality (the functionality corresponding to IBE), in the non-programmable random oracle model. As the proof closely mirrors the argument of Nielsen [Nie02] for non-interactive non-committing encryption, we give only a high-level overview of the proof.

We note that our impossibility result in fact holds for much less stringent formulations of simulation security for functional encryption. In particular, we consider the following weaker version of our main definition:

Definition 5. *An FE scheme $\mathcal{E}$ is weakly simulation-secure if for any (oracle) PPT algorithms $Message$ and Adv, there exists an (oracle) PPT algorithm Sim such that we have that the following two distribution ensembles (over the security parameter λ) are computationally indistinguishable:*

Real Distribution:

1. $(pp, mk) \leftarrow \text{setup}(1^\lambda)$
2. $(\vec{x}, \tau) \leftarrow Message(1^\lambda)$
3. $\vec{c} \leftarrow \text{enc}(pp, \vec{x})$
4. $\alpha \leftarrow Adv^{\text{keygen}(mk,\cdot)}(pp, \vec{c}, \tau)$
5. *Let $y_1, \ldots, y_\ell$ be the queries to* keygen *made by Adv in the previous steps.*
6. *Output* $(\vec{x}, \tau, \alpha, y_1, \ldots, y_\ell)$

Ideal Distribution:

1. $(\vec{x}, \tau) \leftarrow Message(1^\lambda)$
2. $\alpha \leftarrow Sim^{F(\cdot,\vec{x})}(1^\lambda, \tau, F(\epsilon, \vec{x}))$
3. *Let $y_1, \ldots, y_\ell$ be the queries to F made by Sim in the previous step.*
4. *Output* $(\vec{x}, \tau, \alpha, y_1, \ldots, y_\ell)$

We note another weakening of the definition above would be to have the distributions output the queries $y_1, \ldots, y_\ell$ as an unordered set, instead of an ordered tuple. Our impossibility proof can be extended to this weakening as well. We now sketch the proof of the following theorem.

Theorem 2. *Let F be the functionality for IBE. There does not exist any weakly simulation-secure FE scheme for F in the non-programmable random oracle model.*

Brief Proof Sketch. The overall idea of this proof is almost identical to the impossibility proof of Nielsen [Nie02] for non-interactive non-committing encryption. Let H represent the random oracle. Consider the following concrete adversary algorithms:

$Message(1^\lambda)$ works as follows: Let len_{sk} be the maximum bit length produced by the keygen algorithm for the key 0 for security parameter λ. Then the vector $\vec{x}$ consists of the following elements: for $i = 1, \ldots, len_{sk} + \lambda$, the element $(r_i, 0)$ where r_i is a randomly and independently chosen bit for each i. The value τ is empty.

$Adv^{\text{keygen}(\text{mk},\cdot)}(\text{pp}, \vec{c}, \tau)$ works as follows: Call the random oracle H on the input $(\text{pp}, \vec{c})$ to obtain a string w of length λ. Now request the secret key for the identity (w) first, and then for the identity 0. Use

the key for identity 0 to decrypt the entire ciphertext. Output a full transcript of the entire computation done by Adv, including all calls to the random oracle and the interaction with the keygen oracle.

Now consider what Sim must do in order to output a distribution indistinguishable from the real interaction. Because Adv only makes a single key query of the form (w), it is the case that Sim must make exactly one query – its first query – to F of this form as well. Furthermore, the distinguisher can check if this w is the output of H applied to some string of the form $(\text{pp}, \vec{c})$. Thus, the simulator must perform this query to H before making any queries to F. The simulator at this point has no information whatsoever about the plaintexts r_i (which is only revealed when the simulator queries F for identity 0 afterwards). Thus, this fixed string $z = (\text{pp}, \vec{c})$ has the (impossible) property that after receiving only len_{sk} bits of information, it can deterministically "decode" z to be a an arbitrary string of length $len_{sk} + \lambda$. □

We remark that the proof above made use of the fact that the simulator's queries to F are recorded *in order*. However, we note that the same impossibility result would hold even if the security definition only recorded the *unordered set* of queries to F, but using a slightly more involved adversary and message distribution. Roughly speaking, the only identities in the system would be of the form $(i, 0)$ and $(i, 1)$ for $i = 1, \ldots, \lambda$, and the messages to be encrypted would be random long messages for each identity. The adversary would apply the random oracle to $(\text{pp}, \vec{c})$ to obtain a single string w of length λ exactly as above, but it would now use this string to obtain keys (i, w_i) for $i = 1, \ldots, \lambda$. The argument would now proceed by looking at the point when the simulator has made at least $\lambda/2$ queries to F. By now with overwhelming probability, a single query $(\text{pp}, \vec{c})$ to H would be compatible with these queries, and that could be used to define the "impossible string" needed above.

5.2 A simulation-based brute force scheme

We now consider FE schemes that are simulation-secure in the random oracle model (where the scheme algorithms and the $Message$ and Adv algorithms all have oracle access to a random oracle, but the simulator algorithms can emulate the random oracle itself). We note that this is the standard formulation of the random oracle model, more recently called the "full" or "programmable" random oracle model.

The modified "brute-force" construction. We first consider the following slight modification of the brute-force construction given earlier. The modification just uses the random oracle to randomly mask the output values of the function.

We will make use of a random oracle $H : \{0,1\}^* \to \{0,1\}$. Note that we will abuse notation and also write $H(x)$ to produce strings of arbitrary length (which will be clear from context). This can be accomplished by interpreting $H(x)$ to mean the concatenation of $H((x,1)), \ldots, H((x,\ell))$ to produce strings of length ℓ.

Recall that we write $s = |K| - 1$ and $K = \{\epsilon, k_1, \ldots, k_s\}$. The brute force FE scheme realizing F uses a semantically secure public-key encryption scheme (G, E, D), and works as follows:

- setup(1^λ): for $i = 1, \ldots, s$ run $(\text{pp}_i, \text{mk}_i) \leftarrow G(1^\lambda)$.

  $$\text{output:} \quad \text{pp} := (\text{pp}_1, \ldots, \text{pp}_s) \quad \text{and} \quad \text{mk} := (\text{mk}_1, \ldots, \text{mk}_s)$$

- keygen(mk, k_i): output $\text{sk}_i := \text{mk}_i$.

- enc(pp, x): choose random values $r_1, \ldots, r_s \in_R \{0,1\}^\lambda$.
  output $\vec{c} := \big(F(\epsilon, x),\ E\big(\text{pp}_1, r_1\big), H(r_1) \oplus F(k_1, x),\ \ldots,\ E\big(\text{pp}_s, r_s\big), H(r_s) \oplus F(k_s, x) \big)$.

- $\mathrm{dec}(\mathrm{sk}_i, \vec{c})$: output c_0 if $\mathrm{sk}_i = \epsilon$, and output $H\big(D(\mathrm{sk}_i, c_{2i-1})\big) \oplus c_{2i}$ otherwise.

A proof sketch of the following theorem can be found in Appendix A.

Theorem 3. *Let F be a functionality that reveals functional bit lengths. If (G, E, D) is a semantically secure public-key encryption scheme then the modified brute force FE system above implementing F is simulation-secure in the random oracle model.*

5.3 An equivalence for public index schemes

We show that any predicate encryption system with public index that is secure under the game-based Definition 3 also satisfies the simulation based Definition 4 in the random oracle model. This result shows equivalence (in the random oracle model) for the large class of public index schemes including various forms of Attribute-Based encryption.

Let $\mathcal{E} := (\mathrm{setup}, \mathrm{keygen}, \mathrm{enc}, \mathrm{dec})$ be an FE predicate encryption system with public index for predicate $P : K \times I \to \{0,1\}$. We convert the system into a scheme $\mathcal{E}_H := (\mathrm{setup}, \mathrm{keygen}, \mathrm{enc}_H, \mathrm{dec}_H)$ where encryption is done using a random oracle H:

- $\mathrm{enc}_H(\mathrm{pp}, (\mathsf{ind}, m))$: choose a random value $r \in \{0,1\}^\lambda$ and output

$$c := \big(\ \mathrm{enc}(\mathrm{pp}, (\mathsf{ind}, r)),\ \ H(r) \oplus m\ \big)\ .$$

- $\mathrm{dec}_H(\mathrm{sk}, (c_1, c_2))$: if $\mathrm{dec}(\mathrm{sk}, c_1) = \perp$ output $\perp$, otherwise output $\mathrm{dec}(\mathrm{sk}, c_1) \oplus c_2$.

The following theorem shows that this construction is simulation secure.

Theorem 4. *If the system $\mathcal{E}$ is game-secure (Definition 3) then $\mathcal{E}_H$ is simulation secure (Definition 4) in the random oracle model.*

Proof Sketch. We construct the universal simulators Sim_1, $SimO$, and Sim_2 needed for simulation security. These algorithms work as follows:

- $Sim_1(1^\lambda)$ simply executes $\mathrm{setup}(1^\lambda)$ to obtain pp and mk. It outputs pp unmodified, and outputs $\sigma = (\mathrm{mk}, O, \kappa)$, where O and κ are empty lists. This list O will keep track of the simulated random oracle, and κ will keep track of key queries.

- $SimO(\cdot)[[\sigma]]$ works as follows: It responds to random oracle queries and keygen queries the adversary $Message$ makes as follows:

  - **Random Oracle Queries:** On query q, the simulator first checks to see if a pair (q, y) already exists in the list O. If so, it provides y as the response to the adversary's query. If not, the simulator chooses a fresh random string y, adds the pair (q, y) to the list O, and provides y as the response to the adversary's query. This list O is updated in the state variable σ.
  - **Key Queries:** When the adversary asks for the key k, the simulator sends the secret key $\mathrm{sk} \leftarrow \mathrm{keygen}(\mathrm{mk}, k)$ to the adversary. The simulator adds k to the list κ. This list κ is updated in the state variable σ.

- $Sim_2^{F(\vec{x},\cdot),\ Adv^\circ(\mathrm{pp},\cdot,\tau)}(\sigma, F(\vec{x}, \epsilon))$ works as follows:

1. The algorithm begins by preparing a "fake" vector of ciphertexts as follows: Let n be the number of elements in $\vec{x}$ and let $\mathsf{ind}_1, \ldots, \mathsf{ind}_n$ be the indices in $\vec{x}$. Sim_2 obtains n and these indices by querying its F oracle at $F(\epsilon, \vec{x})$.

   Now, for $i = 1 \ldots n$, it chooses random strings $r_1, \ldots, r_n$ and $R_1, \ldots R_n$, and creates the ciphertext components

   $$c_{i,1} := \mathsf{enc}\big(\mathsf{pp}, (\mathsf{ind}_i, r_i)\big) \quad \text{and} \quad c_{i,2} = R_i \qquad \text{for } i = 1, \ldots, n.$$

   Let $\vec{c}$ be the vector of n ciphertexts $\vec{c} := (c_{i,1}, c_{i,2})_{i=1,\ldots,n}$.

2. For each key k in the list κ of keys already queried, the simulator does the following:
   (1) it invokes the F oracle and obtains $F(k, \vec{x}) = (z_1, \ldots, z_n)$,
   (2) for $i = 1, \ldots, n$ if $z_i \neq \perp$ it adds the pair $(r_i, R_i \oplus z_i)$ to the list O. If any of these r_i values were already in the list O, the simulation aborts.

3. Then it invokes $Adv(\mathsf{pp}, \vec{c}, \tau)$ using this "fake" ciphertext vector $\vec{c}$ created above.

4. It now monitors which random oracle queries and keygen queries the adversary Adv makes. It responds to these queries as follows:
   - **Random Oracle Queries:** On query q, the simulator first checks to see if a pair (q, y) already exists in the list O. If so, it provides y as the response to the adversary's query. If not, the simulator chooses a fresh random string y, adds the pair (q, y) to the list O, and provides y as the response to the adversary's query.
   - **Key Queries:** If the adversary asks for the key k, then the simulator invokes the F oracle and obtains $F(k, \vec{x}) = (z_1, \ldots, z_n)$. For $i = 1, \ldots, n$ if $z_i \neq \perp$ it adds the pair $(r_i,\ R_i \oplus z_i)$ to the list O. If for any i there is already a pair (r_i, R) in the list O with $R \neq R_i \oplus z_i$ then the simulation aborts. Finally, it sends the secret key $\mathsf{sk} \leftarrow \mathsf{keygen}(\mathrm{mk}, k)$ to the adversary. It is easy to confirm that the decryption procedure will work as it should after we have modified the random oracle as detailed above.

5. When the adversary terminates and outputs α, then the simulator outputs this α as well, finishing the simulation.

The same argument as in the proof of Theorem 3 shows that the simulator aborts with negligible probability and that the distribution generated by these simulators is statistically close to the real distribution. In particular, the negligible probability of abort follows from the game-based security of $\mathcal{E}$, since game-based security implies one-way security for encrypting random values, which implies that the adversary is extremely unlikely to query the random oracle on the r_i values prior to obtaining a secret key that can open the i'th ciphertext. □

Other Simulation-Secure Functional Encryption Schemes. Since the above equivalence only applies to public index schemes, an interesting question is whether we can achieve simulation security for more general systems. Intuitively this is more challenging, since it goes beyond just hiding a payload, to "hiding a computation" and is arguably closer to our counter example of Section 4.2.

In Appendix B we prove the simulation security of the Boneh-Franklin construction for the anonymous IBE functionality. An interesting direction is to prove simulation security for systems with more functionality. One challenge is that it is not completely clear how to apply the random oracle heuristic to these systems, as the correctness of such schemes typically relies on structure that a hash function might break.

6 Extending Functional Encryption

In this work, we focus on the "core" case of functional encryption. However, there are multiple ways to extend the concept. We briefly outline these here. We hope future work will develop these extensions and give precise definitions of security and constructions.

Delegation. Delegation is the ability of an algorithm to transform a key k in a functional encryption system to another key k'. For example, one might want to share the ability to decrypt all messages of a certain subject to another user. Typically, we think of the resulting key k' as being more restricted than the source key k. We observe that the set of allowed delegations must respect the security definition of the system. Delegation in functional encryption systems is typically associated with Hierarchical Identity-Based Encryption [HL02, GS02], but was also considered in Attribute-Based Encryption [GPSW06] and other predicate encryption systems [SW08].

Encryption over Multiple Parameters and Multiple Systems. Our functional encryption systems allow for functionalities $F : K \times X \to \{0,1\}^*$ that take in a single key and plaintext as inputs. However, we could extend our system to allows for functionalities that take in multiple keys $F : (K_1, \ldots, K_n) \times X \to \{0,1\}^*$. This can be useful in applications where we want users to combine their capabilities in a specified manner or when one of the keys can represent an event such as a certain time period arriving, or publication of a revocation list [BGK08].Another interesting direction is to allow for a functional encryption system that operates over multiple ciphertexts.

Taking things further we could consider an encryption system where encryption takes in multiple public parameters each from *different* authorities and where the functionality is evaluated over private keys generated by different master secret keys. One notable application of this is Attribute-Based Encryption over multi-authorities [Cha07, CC09].

Hiding Information about capabilities of the key. One consistent feature of all systems is that there do not exist any security notions about the attacker's inability to distinguish what type of key k he is given a secret key for. A natural reason for this is that in a public key system, he can distinguish whether he has the capability for k_0, k_1 by simply encrypting an $x \in X$ such that $F(k_0, x) \neq F(k_1, x)$. However, one might try to consider such a problem when encryption is not public key [SSW09, BIP10].

7 Future directions in functional encryption

The results to date scratch the surface of functional encryption and only implement relatively simple functionalities. Here we list a few fascinating open problems that remain.

- The grand challenge is to construct a secure functional encryption scheme for all polynomial-time functionalities. A more modest goal is to do the same for predicate encryption for all polynomial-time predicates. Currently, the best we can do is predicates defined by inner products [KSW08]. The inner product construction uses bilinear maps and our inability to move beyond inner products is due to the "bi" in bilinear maps. Other tools, perhaps borrowing from fully homomorphic encryption [Gen09], may lead to a more general class of predicates.

- If not all polynomial time functionalities, can we realize complex interesting ones such as data-mining functionalities? That is, can we build a secret key that given an encrypted data set will produce a cleartext model (e.g. a decision tree) for the data set, but reveal nothing else about the data? Nothing in this vain is currently known.
- Are there black box separations between different functionalities? Currently, the only result in this direction separates IBE from public-key encryption [BPR+08]. Is there a generic separation result that separates any two functionalities that are not trivially implied one by the other?

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (h)ibe in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

[ABC+08] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptology*, 21(3):350–391, 2008.

[AI09] Nuttapong Attrapadung and Hideki Imai. Conjunctive broadcast and attribute-based encryption. In *Pairing*, pages 248–265, 2009.

[AL10] Nuttapong Attrapadung and Benoît Libert. Functional encryption for inner product: Achieving constant-size ciphertexts with adaptive security or support for negation. In *Public Key Cryptography*, pages 384–402, 2010.

[BB04a] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

[BB04b] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.

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

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003. extended abstract in Crypto 2001.

[BGK08] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based encryption with efficient revocation. In *ACM Conference on Computer and Communications Security*, pages 417–426, 2008.

[BH08] Dan Boneh and Mike Hamburg. Generalized identity-based and broadcast encryption schemes. In *Proc. of Asiacrypt*, pages 455–470, 2008.

[BIP10] Carlo Blundo, Vincenzo Iovino, and Giuseppe Persiano. Predicate encryption with partial public keys. Cryptology ePrint Archive, Report 2010/476, 2010. http://eprint.iacr.org/.

[BPR+08] Dan Boneh, Periklis A. Papakonstantinou, Charles Rackoff, Yevgeniy Vahlis, and Brent Waters. On the impossibility of basing identity based encryption on trapdoor permutations. In *FOCS*, pages 283–292, 2008.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[BW06] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO*, pages 290–307, 2006.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[CC09] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *ACM Conference on Computer and Communications Security*, pages 121–130, 2009.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC 1996*, pages 639–648, 1996.

[Cha07] Melissa Chase. Multi-authority attribute based encryption. In *TCC*, pages 515–534, 2007.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552, 2010.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[Del07] Cécile Delerablée. Identity-based broadcast encryption with constant size ciphertexts and private keys. In *ASIACRYPT*, pages 200–215, 2007.

[DH76a] Whitfield Diffie and Martin E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, pages 109–112, 1976.

[DH76b] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[DPP07] Cécile Delerablée, Pascal Paillier, and David Pointcheval. Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In *Pairing*, pages 39–59, 2007.

[Gen06] Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[GJPS08] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In *ICALP (2)*, pages 579–591, 2008.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Jour. of Computer and System Science*, 28(2):270–299, 1984.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *STOC*, pages 291–304, 1985.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *FOCS*, pages 174–187, 1986.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GS02] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In *EUROCRYPT*, pages 171–188, 2009.

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *EUROCRYPT*, pages 466–481, 2002.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Proc. of CRYPTO 2002*, pages 111–126, 2002.

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

[OSW07] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *ACM Conference on Computer and Communications Security*, pages 195–203, 2007.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, pages 214–231, 2009.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.

[SBC+07] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy*, pages 350–364, 2007.

[SF07] Ryuichi Sakai and Jun Furukawa. Identity-based broadcast encryption. Cryptology ePrint Archive, Report 2007/217, 2007. http://eprint.iacr.org/.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *TCC*, pages 457–473, 2009.

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

[SW08] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In *ICALP (2)*, pages 560–578, 2008.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

[Wat08] Brent Waters. Functional encryption:beyond public key cryptography. Power Point Presentation, 2008. http://userweb.cs.utexas.edu/~bwaters/presentations/files/functional.ppt.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. PKC, 2011.

A Proof of simulation security for modified brute-force scheme

In this section, we provide a proof sketch for Theorem 3.

Proof Sketch. We shall construct the universal simulators Sim_1, $SimO$, and Sim_2 that are demanded by the security definition. These algorithms work as follows:

- $Sim_1(1^\lambda)$ simply executes setup(1^λ) to obtain pp and mk. It outputs pp unmodified, and outputs $\sigma = (\text{mk}, O, \kappa)$, where O and κ are empty lists. This list O will keep track of the simulated random oracle, and κ will keep track of key queries.
- $SimO(\cdot)[[\sigma]]$ works as follows:
  1. It now responds to random oracle queries and keygen queries the adversary $Message$ makes. It responds to these queries as follows:
     - **Random Oracle Queries:** On query q, the simulator first checks to see if a pair (q, y) already exists in the list O. If so, it provides y as the response to the adversary's query. If not, the simulator chooses a fresh random string y, adds the pair (q, y) to the list O, and provides y as the response to the adversary's query. This list O is updated in the state variable σ.
     - **Key Queries:** If the adversary asks for the key k_i, then the simulator sends the secret key mk_i to the adversary. The simulator adds k_i to the list κ. This list κ is updated in the state variable σ.

- $Sim_2^{F(\vec{x},\cdot),\ Adv(\text{pp},\cdot,\tau)}(\sigma, F(\vec{x},\epsilon))$ works as follows:

  1. The next simulation algorithms begins by preparing a "fake" vector of ciphertexts as follows: Let m be the number of elements in $\vec{x}$ (this number is known from the input $F(\vec{x},\epsilon)$ that is provided to Sim_2). For $i = 1 \ldots m$, it chooses random strings $r_{i,1}, \ldots, r_{i,s}$ and $R_{i,1}, \ldots R_{i,s}$, and for $j = 1 \ldots s$, it creates the ciphertext components $c_{i,2j-1} = E\big(\text{pp}_j, r_{i,j}\big)$ and $c_{i,2j} = R_{i,j}$. If the list O already contains any oracle queries to any of the randomly chosen $r_{i,j}$'s, the simulator aborts.
  2. For all keys k_i in the list κ of keys already queried, the simulator invokes the F oracle and obtains $F(k_i, \vec{x}) = (z_1, \ldots, z_m)$. It now adds the pairs $(r_{i,1}, R_{i,1} \oplus z_1), \ldots, (r_{i,m}, R_{i,m} \oplus z_m)$ to the list O. If any of these $r_{i,j}$ values were already in the list O, the simulation aborts.
  3. Then it invokes $Adv(\text{pp}, \vec{c}, \tau)$ using this "fake" ciphertext as created above.
  4. It now monitors which random oracle queries and keygen queries the adversary Adv makes. It responds to these queries as follows:
     - **Random Oracle Queries:** On query q, the simulator first checks to see if a pair (q, y) already exists in the list O. If so, it provides y as the response to the adversary's query. If not, then it checks to see if q is equal to any of the random $r_{i,j}$ values chosen above. If so, the simulator aborts. (We will later argue that the probability of such an abort is negligible.) Finally if none of the conditions apply, the simulator chooses a fresh random string y, adds the pair (q, y) to the list O, and provides y as the response to the adversary's query.
     - **Key Queries:** If the adversary asks for the key k_i, then the simulator invokes the F oracle and obtains $F(k_i, \vec{x}) = (z_1, \ldots, z_m)$. It now adds the pairs $(r_{i,1}, R_{i,1} \oplus z_1), \ldots, (r_{i,m}, R_{i,m} \oplus z_m)$ to the list O. Finally, it sends the secret key mk_i to the adversary. Note: it is easy to confirm that the decryption procedure will work as it should after we have modified the random oracle as detailed above.
  5. When the adversary terminates and outputs α, then the simulator outputs this α as well, finishing the simulation.

We observe that if the above simulation only aborts with negligible probability, then the ideal distribution is statistically close to the real distribution, since (except for the abort condition) the simulation above behaves exactly as the real execution.

We now argue that the simulation only aborts with negligible probability. Suppose not. This means that with noticeable probability δ, the adversary queries the random oracle for some $r_{i,j}$ value before asking for the key k_i. We will use this to break the one-wayness security of the underlying public-key encryption scheme.

Suppose we are given externally the public key pk for an encryption scheme and a ciphertext $C = E(\text{pk}, r)$ for a random value $r \in \{0,1\}^\lambda$. We will construct an attacker that outputs r with probability at least δ/sM^2, where M is some polynomial upper bound on m and the number of random oracle queries that the adversary can make. The attacker runs the simulation above, however it chooses $i \in [1, s]$ and $j \in [1, M]$ at random ahead of time, and it replaces pp_i with pk and $r_{i,j}$ with r and $c_{i,2j-1}$ with C (if $j > m$, then the attacker aborts). The attacker continues the simulation until the attacker finishes or until the attacker asks for key k_i. At this point, it randomly picks one of the queries q made by the adversary so far, and outputs this query as its guess for r. This attack will succeed with probability at least δ/sM^2 by construction, which contradicts the security of the public-key encryption scheme. □

B The Boneh-Franklin [BF03] Anonymous IBE

We now show a proof of security for an anonymous-IBE functionality in our simulation definition using a slightly modified version of the Boneh-Franklin IBE. This is the first example of non public index scheme proven secure under such a definition. We use the functional encryption formulation given in Section 3.

The Boneh-Franklin IBE makes use of groups $\mathbb{G}$ that have an efficiently computable bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We refer the reader to [BF03, BB04a] for further discussion on such groups. For simplicity we will assume all messages to be of length λ.

- setup(1^λ): The setup algorithm chooses a prime p of length λ and creates a bilinear group $\mathbb{G}$ with generator g of order p. It then picks a secret exponent $a \in \mathbb{Z}_p$. The parameters are. In addition, it defines a hash functions $T : \{0,1\}^* \to \mathbb{G}$ and $H : \mathbb{G}_T \to \{0,1\}^{2\cdot\lambda}$ that we will model as random oracles. For simplicity we will assume that messages to be encrypted are of the length λ.

  $$\text{output:} \qquad \text{pp} := (\text{Description of } \mathbb{G} \text{ and } T, H\ , g, g^a) \qquad \text{and} \qquad \text{mk} := a$$

- keygen(mk, k): output $\text{sk}_k := T(k)$.

- enc$(\text{pp}, x = (\mathsf{ind}, m))$: choose random value $s, \in \mathbb{Z}_p, r \in_R \{0,1\}^\lambda$.
  compute $y = e(T(\mathsf{ind}), g^a)^s = e(T(\mathsf{ind}), g)^{as}$
  output $\vec{c} := \big(c_1 = g^s, c_2 = H(y) \oplus 0^\lambda | m\big)$.

- dec$(\text{sk}_k, \vec{c})$: compute $(z_1 \in \{0,1\}^\lambda, z_2 \in \{0,1\}^\lambda) = H(e(\text{sk}_k, c_1)) \oplus c_2$
  output $\perp$ if $z_1 \neq 0^\lambda$ and z_2 otherwise.

Theorem 5. *Let F be the anonymous IBE functionality. If the modified Boneh-Franklin construction is a secure anonymous IBE system under our game based Definition 3, then it is secure under our simulation Definition 4.*

The proof of our simulation relies on the fact that no information about *either* the ciphertext identity ind or the message m is present until the attacker calls the random oracle H on the randomness r. Once, an attacker makes a private key query for an identity k, the simulator can then program the random oracle to make any ciphertext for k appear correctly. This simulation is perfect unless an attacker queries H on r for a given ciphertext before learning the private key for its identity. We can then show that the game-based security of Boneh-Franklin implies this will happen with negligible probability.

Proof Sketch. We shall construct the universal simulators Sim_1, $SimO$, and Sim_2 that are demanded by the security definition. These algorithms work as follows:

- $Sim_1(1^\lambda)$ simply executes setup(1^λ) to obtain pp and mk. It outputs pp unmodified, and outputs $\sigma = (\text{mk}, O, O_T, \kappa)$, where O, O_T and κ are empty lists. This lists O, O_T will keep track of the simulated random oracles, and κ will keep track of key queries.

- $SimO(\cdot)[[\sigma]]$ works as follows:

  1. It now responds to random oracle queries and keygen queries the adversary Message makes. It responds to these queries as follows:

- **Random Oracle Queries to H:** On query q, the simulator first checks to see if a pair (q, y) already exists in the list O. If so, it provides y as the response to the adversary's query. If not, the simulator chooses a fresh random string y, adds the pair (q, y) to the list O, and provides y as the response to the adversary's query. This list O is updated in the state variable σ.
- **Random Oracle Queries to T:** The simulator will maintain a separate list O_T of queries to T. If a query has already been made it will return that value, other wise it picks a random element in $\mathbb{G}$ (the group chosen in setup) and records the input/output pair.
- **Key Queries:** If the adversary asks for the key k_i, then the simulator sends the secret key mk_i to the adversary. The simulator adds k_i to the list κ. This list κ is updated in the state variable σ.

- $Sim_2^{F(\vec{x}=(x_1,\ldots,x_\ell),\cdot),\ Adv(\text{pp},\cdot,\tau)}(\sigma, F(\vec{x}, \epsilon))$ works as follows:

  1. Then, it prepares a "fake" vector of ciphertexts as follows: Let ℓ be the number of elements in $\vec{x}$ (this number is known from the input $F(\vec{x}, \epsilon)$ that is provided to Sim_2). For $j = 1 \ldots \ell$, it chooses random strings $s_i j \in \mathbb{Z}_p$ and R_j. It then creates the ciphertext components $c_{j,1} = g^{s_j}$ and $c_{j,2} = R_j$.
  2. For each j from 1 to ℓ and each key k_i in the list κ the simulator calls the F oracle and obtains $F(k_i, x_j)$. For any $j \in [1, \ell]$ if there exists an i such that $F(k_i, x_j) = m_j \neq \perp$, it sets $z_j = m_j$ and sets $w_j = k_i$. Otherwise, it sets $z_i = \perp$. Intuitively, this is is denoting that the j-th ciphertext is set an encryption of message z_j to identity w_j. If $z_j = \perp$ this indicates, that no key has yet been queried that matches the ciphertext. We note that the semantics of the anon-IBE scheme mean that there will be no conflicts in setting z_j, since the only possible outputs from the oracle are $\perp$ and a unique message m_j.

     For each j where $z_j \neq \perp$ it adds the pair $(e(g, T(w_j))^{s_j a}, R_j \oplus 0^\lambda | z_j)$ to the list O, where key a is the master secret from the setup. If any of these $e(g, T(w_j))^{s_j a}$ values were already in the list O, the simulation aborts.
  3. Then it invokes $Adv(\text{pp}, \vec{c}, \tau)$ using this "fake" ciphertext as created above.
  4. It now monitors which random oracle queries and keygen queries the adversary Adv makes. It responds to these queries as follows:
     - **Key Queries:** If the adversary asks for the key k_i, the simulator first checks if the key has already been queried for and returns back the same key in that case. This is suitable since keygen is deterministic in the Boneh-Franklin scheme.
       Otherwise it runs $keygen(k_i, \text{mk})$. Then it tests for all j where $z_j = \perp$ if $F(k_i, x_j) = m_j \neq \perp$. If so for each of these it updates $z_j = m_j$ and $w_j = k_i$. Then it adds the pair $(e(g, T(w_j))^{s_j a}, R_j \oplus 0^\lambda | z_j)$ to the oracle. Finally, it sends the secret key mk_i to the adversary. Note: it is easy to confirm that the decryption procedure will work as it should after we have modified the random oracle as detailed above.
     - **Random Oracle Queries:** On query q, to the oracle H the simulator first checks to see if a pair (q, y) already exists in the list O. If so, it provides y as the response to the adversary's query. Otherwise, the simulator checks if it must abort.
       Let $V = \{v_k\}$ be the set of queries (of identities) in the list O_T, that were queried on, but that there was no private key asked for in the list κ. For every $v_k \in V$ and every j where $z_j = \perp$, the simulator tests is the query $q = e(T(v_k), g)^{a s_j}$. If this occurs the simulation

aborts. The reason for aborting, is that answering this query would take away the simulator's ability to program the random oracle at that point further in the simulation. We will argue this abort event happens with negligible probability.
Finally, if neither of these conditions occur the simulator chooses a fresh random string y, adds the pair (q, y) to the list O, and provides y as the response to the adversary's query. For queries to T, the simulator continues the same simulation from the previous algorithm.

5. When the adversary terminates and outputs α, then the simulator outputs this α as well, finishing the simulation.

We observe that if the above simulation only aborts with negligible probability, then the ideal distribution is statistically close to the real distribution, since (except for the abort condition) the simulation above behaves exactly as the real execution.

We now sketch why the abort condition will be given with negligible probability. Let M be some polynomial upper bound on the ℓ and the number of random oracle queries made. In Sim_2 there are two places, Steps 2 and 4, where the simulator may abort. At step 2, each ciphertext is created with a fresh $s_j \in \mathbb{Z}_p$, thus the probability of the attacker querying $e(g, T(w_j))^{s_j a}$ for a particular j before its creation is a most M/p and for any j is bounded by M^2/p, which is negligible.

We argue that any attacker that would cause an abort to happen in Step 4 with non-negligible probability u , can break the computation Bilinear Diffie-Hellman (BDH) problem with non-negligible probability. [9] Suppose there did exist such an attacker, then we build a reduction. Our reduction first takes in a BDH challenge $(\mathbb{G}, p, (g, g^a, g^b, g^c))$. The attacker first chooses uniformly at random $i^*, j^*, t^* \in [1, M]$, guessing that the attacker would have caused an in the simulation abort due to the i^* query to H, the j^*-th ciphertext and the t^*-th query to T. Since the number of ciphertexts and queries to the oracle are both bounded by M, the guess will be correct with probability at least u/M^3, which is non-negligible.

The reduction begins by running the attacker algorithms in the real game. It publishes $\mathbb{G}, g, g^a$ as the public parameters. For every query to H, until the i^* one it outputs a random value and keeps track of it in a table. For each query to T, it first checks if it has already answered it and if so repeats the value. Otherwise, if it is the $t \neq t^*$ it chooses a random $y_t \in \mathbb{Z}_p$ as the response. For t^* it replies with g^b. For the $j \neq j^*$ ciphertext it simply encrypts x_j according to the algorithm. For $j = j^*$ it sets $c_{j^*,1} = g^c$ and $c_{j^*,2}$ to a random string. To answer key queries for identity k, the algorithm first identifies the index t for which the attacker called $T(k)$ (if none exists, the reduction can simply call $T(k)$ itself). For any index $t \neq t^*$ the reduction replies with $(g^a)^{y_t}$, which is distributed as a well formed key. If $t = t^*$, the reduction halts and quits. In this case the guess of t^* was wrong since the abort condition of 4 only happens when the identity for t^* was not already for. Finally, whenever the i^* query to H occurs the reduction outputs this as its solution to the BDH problem.

We first observe that if the reduction did not halt, then the view is distributed equally to the real scheme. Finally, the condition that the attacker would cause an abort on the i^*-th query to H in Step 4 due to the j^* ciphertext and t^* query to T is implies that it must have submitted $e(g, g)^{abc}$ which is a solution to the BDH problem.

□

[9] We refer the reader to [BB04a] for the description of the BDH problem in our notation.