Functional Encryption for Regular Languages

Brent Waters[1] *

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

Abstract. We provide a functional encryption system that supports functionality for regular languages. In our system a secret key is associated with a Deterministic Finite Automata (DFA) M. A ciphertext CT encrypts a message m and is associated with an *arbitrary length* string w. A user is able to decrypt the ciphertext CT if and only if the DFA M associated with his private key accepts the string w.
Compared with other known functional encryption systems, this is the first system where the functionality is capable of recognizing an unbounded language. For example, in (Key-Policy) Attribute-Based Encryption (ABE) a private key SK is associated with a single boolean formula ϕ which operates over a *fixed* number of boolean variables from the ciphertext. In contrast, in our system a DFA M will meaningfully operate over an arbitrary length input w.
We propose a system that utilizes bilinear groups. Our solution is a "public index" system, where the message m is hidden, but the string w is not. We prove security in the selective model under a variant of the decision ℓ-Bilinear Diffie-Hellman Exponent (BDHE) assumption that we call the decision ℓ-Expanded BDHE problem.

1 Introduction

Functional encryption is an emerging new way of viewing encryption. Instead of encrypting data to a particular user, an encryptor will encrypt data under the public parameters of a system. Users in the system will obtain a secret key SK (issued from some authority) that is associated with a value k . What a user learns about the data depends on the systems functionality and their key value k.

Perhaps the most well known and representative functional encryption system is Attribute-Based Encryption (ABE) [25, 18]. In a (Key-Policy) ABE system, a key is associated with a *single* boolean formula ϕ and a ciphertext is associated with a pair of a message m and variables $\boldsymbol{x}$. A user can decrypt and learn m if and only if $\boldsymbol{x}$ satisfies the formula ϕ. One salient feature of this system is that the formula ϕ only operates over a *fixed* number of variables (i.e., a bounded description from the ciphertext). However, there are many applications where we might want the functionality key to operate over arbitrary sized

* Supported by NSF CNS-0952692, the Alfred P. Sloan Fellowship, and Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

input data. For example, we could imagine a network logging application where an encryption input represents an arbitrary number of events logged. Another example is an encryption of a database of patient data that includes disease history paired with gene sequences where the number of participants is not a priori bounded or known. This restriction on fixed size inputs is not limited to Attribute-Based Encryption; other functional encryption systems such as Spatial Encryption [10, 19] or Inner Product Encryption [21, 23, 24] also operate over fixed size inputs.[1]

Functional Encryption over Arbitrary Size Inputs We initiate the study of functional encryption systems that operate over arbitrary size inputs. We begin with systems that support a regular language functionality. Regular languages provide a natural starting point since they are well studied and are used in various practical applications such as specifying rules in firewall systems.

A general class of applications is where there is a family of documents that has a public (not hidden) template along with some secret data that fits within that template. For instance, one can consider tax returns — which can be arbitrarily long — where the template consists of the deductions claimed and the private data is the amount of these deductions. An auditor might be authorized to read the private data of a return if the deductions match a certain regular expression. Another example is a virus scanner for webpages. Here we might wish the scanner to be able to take a deeper look into pages that match a certain pattern, but have some content remain hidden otherwise.

In general there are multiple sources of data such as video and databases that can be of arbitrary length. While a long term goal is to realize functional encryption for arbitrary programs, regular languages capture interesting properties. For instance, one can check for the presence of substrings or check the parity of data.

Our Contribution We construct what we call a Deterministic Finite Automata (DFA)-based Functional Encryption system. In our system the encryption algorithm takes in a pair of a message m and an arbitrary length string w over a finite alphabet Σ. The key generation algorithm takes as input the description of a DFA $M = (Q, \Sigma, \delta, q_0, F)$ and outputs a secret key. Briefly, Q is a set of states, Σ the alphabet, $\delta : Q \times \Sigma \rightarrow Q$ a next state transition function, $q_0 \in Q$ a unique start state and $F \subseteq Q$ a set of accept states.[2] A user with a secret key for DFA M will be able to decrypt a ciphertext CT associated with w if and only if M accepts the string w. Our system is of the "public index" type [11] (like ABE) in that while the message m is hidden w can be determined from the ciphertext (without a secret key).

Designing an encryption system to support a DFA-type functionality provides a new set of challenges. It is useful to compare them to those in building

[1] While a regular language can be recognized by a family of boolean formulas, a secret key in an ABE system is associated with a single formula.

[2] We give an overview of DFAs in Section 2.

a boolean formula (ABE) type system. In existing ABE systems a ciphertext had to represent the variables in $\boldsymbol{x}$. This generally consisted of either including or excluding a particular ciphertext component depending on whether a corresponding variable was true. In our DFA system we must meaningfully represent an arbitrary length string w. Here representing the order of the symbols comprising w is paramount. In an ABE key generation phase a boolean formula can be mapped closely with the linear secret sharing of the master secret key in the exponent. Similarly, decryption maps closely with the linear reconstruction of these exponents (post pairing with pieces of the ciphertext). Building a DFA description M into a secret key and enforcing its execution during decryption is a more complex task.

We now give a high level overview of some of the main ideas of our construction. We defer several details to Section 3 where the reader will have the benefit of our construction description. Our construction makes use of (prime order) bilinear groups and is focused on three main mechanisms for decryption. We call these the initiation, transition, and completion mechanisms. At a high level the only useful actions a decryptor or attacker should be able to take are by applying them. These mechanisms are realized by how we structure ciphertexts and private keys.

When encrypting a ciphertext for string w of ℓ symbols, the encryptor chooses $\ell + 1$ random exponents $s_0, s_1, \ldots, s_\ell \in \mathbb{Z}_p$ where p is the order of the group. A private key machine $M = (Q, \Sigma, \delta, q_0, F)$ will have $|Q|$ random group elements chosen $D_0, \ldots, D_{|Q|-1}$, where D_x is associated with state q_x.

Suppose a user is trying to decrypt a ciphertext associated with string w with a secret key SK for machine M. Throughout the process of decryption we would like to enforce that the user (or attacker) can only compute $e(g, D_x)^{s_j}$ if the machine M lands on state q_x after reading j symbols of w.

First, the initiation mechanism allows the decryptor to obtain the value $e(g, D_0)^{s_0} \in \mathbb{G}_T$. This initiation mechanism should only be good for computing this value and not useable for other D_i, s_i. Next, the decryptor can repeatedly apply the transition mechnamism. The transition mechanism is structured such that one can compute $(e(g, D_x)^{s_j})^{-1} e(g, D_y)^{s_{j+1}}$ if and only if the $j + 1$-th symbol of w is σ and the transition $\delta(x, \sigma) = y$. Multiplying this into the accumulated value will allow the decryption algorithm to change the accumulated value $e(g, D_x)^{s_j}$ to $e(g, D_y)^{s_{j+1}}$. Thus, mirroring the computation of the DFA M on w. Finally, we will end up with some accumulated value $e(g, D_x)^{s_\ell}$. If the state q_x is an accept state in F, then one can apply the completion mechanism which computes $e(g, g)^{\alpha s_\ell} (e(g, D_x)^{s_\ell})^{-1}$ and this can be multiplied in to simply obtain $e(g, g)^{\alpha s_\ell}$. This is the term used to blind the message.

We prove security in the selective model under a variant of the decision ℓ-Bilinear Diffie-Hellman Exponent (BDHE) assumption that we call the decision ℓ-Expanded BDHE assumption. The parameter ℓ of the assumption depends upon the length of the string w^* associated with the challenge ciphertext. Our proof uses what has been called a partitioning strategy where a string w^* is embedded into the public parameters in such a way that a reduction algorithm can create private keys for any DFA M that does not accept w^*. Our proof will also need to somehow reflect the computation of M on w^* when creating private keys for M. The main reason for applying a parameterized assumption is that our reduction needs to embed an arbitrary size w^* into fixed small sized parameters.

Efficiency We give a brief overview of the computation and storage costs associated with our system. The public parameters contain $5 + |\Sigma|$ group elements, where $|\Sigma|$ is the size of the alphabet. A ciphertext consists of $5+2|w|$ group elements and an encryption costs $5 + 3|w|$ exponentiations, where $|w|$ is the string associated with the ciphertext. A private key consists of $4+3|\mathcal{T}|$ group elements, where $|\mathcal{T}|$ is the number of transitions in the DFA M associated with the key. Finally, a successful decryption requires $4 + 3|w|$ pairing operations.

Backtracking, NFAs and Future Challenges One additional complexity to the prior discussion of mechanisms is that the transition function can be applied in reverse to "backtrack" through the computation. At first this does not seem to be any issue at all since (using our above example) it seems an attacker will only be able to go back from a value $e(g, D_y)^{s_{j+1}}$, representing that he was at state q_y after $j + 1$ symbols, to a prior value $e(g, D_x)^{s_j}$ which represents that earlier the machine was at state q_x after j symbols. However, a twist occurs if there exists another state x' and the transition function $\delta(x', \sigma) = y$, where the $j + 1$-th symbol of w is σ.

In this case the attacker can actually backtrack along the "wrong" transition and compute $e(g, D_{x'})^{s_j}$; falsely representing that after evaluating the first j symbols of w M was in state $q_{x'}$! It turns out that in the case of our DFA system — reflected in our proof— this is not a problem as an attacker can only go backwards so far. When it goes forward again the determinism of M means that we will eventually return to the same state q_y after $j + 1$ symbols.

While this backtracking attack primarily complicates the proof of our DFA-based system, it is indicative that realizing efficient constructions of Nondeterministic Finite Automata (NFA) Functional Encryption systems will likely be difficult. In particular, the most natural extension of our construction to NFAs falls victim to a backtracking attack where an attacker used the inverse transitions and followed by a different forward transitions (as could be allowed by the nondeterminism). Creating efficient NFA constructions is an interesting problem since the best generic way of representing an NFA (or regular expressions) in terms of a DFA requires an exponential blowup in the number of states. However, we suspect that achieving this will be difficult and perhaps related to the difficulty of building efficient ABE systems for circuits.

We also comment that our current proof techniques use both the selective model and a parameterized assumption. An interesting question is if either or both of these properties can be improved using Dual System Encryption [29] proof techniques. While Dual System Encryption was used to prove ABE systems fully secure under static assumptions [22], the core technique encumbered a "one-use" restriction on attributes in formulas which had to be overcome using a certain encoding technique. The encoding bounded at setup the number of times an attribute could be used in a formula. The natural analog of this encoding and its resulting setup bound would negate the motivation of having arbitrary string sizes as the ciphertext input.

A final open challenge is to design a functional encryption system where the DFA M associated with a key is applied directly on encrypted data. That is the string w itself would be hidden and a user would only learn whether M did or did not accept w. Creating such a cryptosystem and moving beyond the public index model in this setting appears challenging.

1.1 Other Related Work

Identity-Based Encryption (IBE), which is one of the most basic forms of functional encryption, was introduced by Shamir [26] in 1984. It wasn't until much later that Boneh and Franklin [9] and Cocks [14] independently proposed IBE systems. There have been multiple IBE constructions using bilinear maps [5, 28, 15] and more recently lattice-based constructions [16, 13, 2, 3].

The beginning of functional encryption can be traced back to early Attribute-Based Encryption systems [25, 18]. There exists a complimentary form of ABE known as Ciphertext-Policy ABE [4, 17, 30] where the secret key is associated with a set of variables and the ciphertext with a boolean formula ϕ. A natural analog in our DFA-based encryption is to associate a DFA M with the ciphertext and a string with a private key.

In the terminology of [11] our scheme works in the public index model since the string w is not hidden. Starting with Anonymous IBE systems [8, 1] systems there have been multiple systems that attempt to hide this information. Some examples include Hidden Vector Encryption [12, 20] and Inner Product functionalities [21, 23, 24] among others.

2 Functional Encryption for DFAs: Definitions and Assumption

We now give a formal definition of our DFA-Based Functional Encryption scheme. In the terminology of Boneh, Sahai, and Waters [11] we give a functional encryption scheme for functionality $F(k, x = (w, m))$ where k is the description of a deterministic finite automata M and x is the the pair of a string w and a message m. The functionality F outputs the encrypted message m if the machine M accepts w and otherwise outputs $\perp$. The functional encryption scheme is of the "public index" type in that the string w is not hidden.

While our system fits within the framework of functional encryption [11] (as sketched above), we choose to present a direct definition for DFA-based functional encryption. We begin by giving a brief overview of DFAs. Then we present our algorithms and game-based security definitions. Finally, we give our bilinear group assumption used to prove security.

2.1 Overview of Deterministic Finite Automata

We give a brief overview of Deterministic Finite Automata (DFA) using terminology and definitions from Sipser [27] and refer the reader to Sipser [27] for further details. A Deterministic Finite Automata M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ in which:

1. Q is a set of states
2. Σ is a finite set of symbols called the alphabet
3. $\delta : Q \times \Sigma \to Q$ is a function known as a transition function
4. $q_0 \in Q$ is called the start state
5. $F \subseteq Q$ is a set of accept states.

For convenience when describing our encryption systems we add the notation of $\mathcal{T}$ as being the set of transitions associated with the function δ, where $t = (x, y, \sigma) \in \mathcal{T}$ iff $\delta(x, \sigma) = y$.

Suppose that $M = (Q, \Sigma, \delta, q_0, F)$. We say that M accepts a string $w = w_1, w_2, \ldots, w_\ell \in \Sigma^*$ if there exists a sequence of states $r_0, r_1, \ldots, r_n \in Q$ where:

1. $r_0 = q_0$
2. For $i = 0$ to $n - 1$ we have $\delta(r_i, w_{i+1}) = r_{i+1}$
3. $r_n \in F$.

We will use the notation $\textsc{Accept}(M, w)$ to denote that the machine M accepts w and $\textsc{Reject}(M, w)$ to denote that M does not accept w. A DFA M recognizes a language L if M accepts all $w \in L$ and rejects all $w \notin L$; such a language is called regular.

2.2 DFA-Based Functional Encryption

We now give our definition of a DFA-based Functional Encryption system. A (Key-Policy) DFA-based encryption scheme consists of four algorithms: Setup, Encrypt, KeyGen, and Decrypt. In addition to the security parameter, the setup algorithm will take as input the description of an alphabet Σ. This alphabet will be used for all strings and machine descriptions in the system.[3] The algorithm descriptions follow:

Setup$(1^n, \Sigma)$ The setup algorithm takes as input the security parameter and the description of a finite alphabet Σ. The alphabet used is shared across the entire system. It outputs the public parameters PP and a master key MSK.

Encrypt(PP$, w, m)$ The encryption algorithm takes as input the public parameters PP, an arbitrary length string $w \in \Sigma^*$, and a message m. It outputs a ciphertext CT.

[3] To fit our system properly in the framework of [11] we would need to have a separate functionality for every alphabet Σ. Then we would have a family of functional encryption systems; one for each alphabet. Here we choose to let Σ be a parameter to the setup algorithm.

Key Generation($\text{MSK}, M = (Q, \mathcal{T}, q_0, F)$*)* The key generation algorithm takes as input the master key MSK and a DFA description M.The description passed does not include the alphabet Σ since it is already determined by the setup algorithm. In addition, we encode the transitions as a set $\mathcal{T}$ of three tuples as described above. The algorithm outputs a private key SK.

Decrypt(SK, CT*).* The decryption algorithm takes as input a secret key SK and ciphertext CT. The algorithm attempts to decrypt and outputs a message m if successful; otherwise, it outputs a special symbol $\perp$.

Correctness Consider all messages m, strings w, and DFA M such that $\textsc{Accept}(M, w)$. If *Encrypt*$(\text{PP}, w, m) \rightarrow \text{CT}$ and *KeyGen*$(\text{MSK}, M) \rightarrow \text{SK}$ where PP, MSK were generated from a call to the setup algorithm, then *Decrypt*(SK, CT) $= m$.

Security Model for DFA-Based Functional Encryption We now describe a game-based security definition for DFA-Based Functional Encryption. As in other functional encryption systems (e.g. [9, 25]), an attacker will be able to query for multiple keys, but not ones that can trivially be used to decrypt a ciphertext. In this case the attacker can repeatedly ask for private keys corresponding any DFA M of his choice, but must encrypt to some string w^* such that every machine M for which a private key was requested for rejects w^*. The security game follows.

Setup. The challenger first runs the setup algorithm and gives the public parameters, PP to the adversary and keeps MSK to itself.
Phase 1. The adversary makes any polynomial number of private keys queries for machine descriptions M of its choice. The challenger returns *KeyGen*(MSK, M).
Challenge. The adversary submits two equal length messages m_0 and m_1. In addition, the adversary gives a challenge string w^* such that for all M requested in Phase 1, $\textsc{Reject}(M, w^*)$. Then the challenger flips a random coin $b \in \{0, 1\}$, and computes *Encrypt*$(\text{PP}, w, m_b) \rightarrow \text{CT}^*$. The challenge ciphertext CT^* is given to the adversary.
Phase 2. Phase 1 is repeated with the restriction that for all M requested $\textsc{Reject}(M, w^*)$.
Guess. The adversary outputs a guess b' of b.

The advantage of an adversary $\mathcal{A}$ in this game is defined as $\Pr[b' = b] - \frac{1}{2}$. We note that the definition can easily be extended to handle chosen-ciphertext attacks by allowing for decryption queries in Phase 1 and Phase 2.

Definition 1 *A DFA-based Functional Encryption system is secure if all polynomial time adversaries have at most a negligible advantage in the above game.*

We say that a system is *selectively* secure if we add an Init stage before setup where the adversary commits to the challenge string w^* and alphabet Σ.

2.3 Expanded ℓ-BDHE Assumption

We define the decision ℓ-Expanded Bilinear Diffie-Hellman Exponent problem as follows. Choose a group $\mathbb{G}$ of prime order $p > 2^n$ for security parameter n. Next choose random $a, b, c_0, \ldots, c_{\ell+1}, d \in {\mathbb{Z}_p}^*$ and a random $g \in \mathbb{G}$. Suppose an adversary is given $\boldsymbol{X}=$

$$g, g^a, g^b, g^{ab/d}, g^{b/d}$$

$$\forall_{i \in [0,2\ell+1], i \neq \ell+1, j \in [0,\ell+1]} \quad g^{a^i s},\ g^{a^i bs/c_j}$$

$$\forall_{i \in [0,\ell+1]}\ g^{a^i b/c_i},\ g^{c_i},\ g^{a^i d},\ g^{abc_i/d},\ g^{bc_i/d}$$

$$\forall_{i \in [0,2\ell+1], j \in [0,\ell+1]} \quad g^{a^i bd/c_j}$$

$$\forall_{i,j \in [0,\ell+1], i \neq j} \quad g^{a^i bc_j/c_i}.$$

Then it must remain hard to distinguish $e(g,g)^{a^{\ell+1}bs} \in \mathbb{G}_T$ from a random element in $\mathbb{G}_T$.

An algorithm $\mathcal{B}$ that outputs $z \in \{0,1\}$ has advantage ϵ in solving decision ℓ-Expanded BDHE in $\mathbb{G}$ if

$$\left| \Pr\left[\mathcal{B}\big(\boldsymbol{X}, T = e(g,g)^{a^{\ell+1}bs}\big) = 0\right] - \Pr\left[\mathcal{B}\big(\boldsymbol{X}, T = R\big) = 0\right] \right| \geq \epsilon$$

Definition 1. *We say that the decision ℓ-Expanded BDHE assumption holds if no polytime algorithm has a non-negligible advantage in solving the decision ℓ-Expanded BDHE problem.*

We give a proof that the assumption holds in the generic group model in the full version of our paper.

3 Construction

We now present our construction of a DFA-based Function Encryption system. We first describe our construction and then provide some additional intuitive discussion. Our formal proof appears in the next section.

3.1 Algorithms

Setup($1^n, \Sigma$) The setup algorithm takes as input a security parameter n and an alphabet Σ. It first chooses a prime $p > 2^n$ and creates a bilinear group $\mathbb{G}$ of prime order p. The algorithm then chooses random group elements $g, z, h_{\text{start}}, h_{\text{end}} \in \mathbb{G}$. In addition, for all $\sigma \in \Sigma$ it chooses random $h_\sigma \in \mathbb{G}$. Finally, an exponent $\alpha \in \mathbb{Z}_p$ is randomly chosen. The master secret key MSK includes $g^{-\alpha}$ along with the public parameters. The public parameters are the description of the group $\mathbb{G}$ and the alphabet Σ along with:

$$e(g,g)^{\alpha}, g, z, h_{\mathrm{start}}, h_{\mathrm{end}},\ \forall_{\sigma\in\Sigma}\ h_{\sigma}.$$

Encrypt($\mathrm{PP}, w=(w_1,\ldots,w_\ell), m$*)* The encryption algorithm takes in the public parameters, an arbitrary length string w of symbols, and a message $m \in \mathbb{G}_T$. Let w_i denote the i-th symbol of w and ℓ denote the length of w. The encryption algorithm chooses random $s_0,\ldots,s_\ell \in \mathbb{Z}_p$ and creates the ciphertext as follows.

First, it sets

$$C_m = m \cdot e(g,g)^{\alpha\cdot s_\ell} \text{ and}$$

$$C_{\mathrm{start}1} = C_{0,1} = g^{s_0},\ C_{\mathrm{start}2} = (h_{\mathrm{start}})^{s_0}$$

Then, for $i=1$ to ℓ it sets:

$$C_{i,1} = g^{s_i}, \quad C_{i,2} = (h_{w_i})^{s_i} z^{s_{i-1}}.$$

Finally, it sets

$$C_{\mathrm{end}1} = C_{\ell,1} = g^{s_\ell},\ C_{\mathrm{end}2} = (h_{\mathrm{end}})^{s_\ell}$$

The output ciphertext is

$$\mathrm{CT} = \big(w, C_m, C_{\mathrm{start}1}, C_{\mathrm{start}2}, (C_{1,1},C_{1,2}),\ldots,(C_{\ell,1},C_{\ell,2}), C_{\mathrm{end}1}, C_{\mathrm{end}2}\big)$$

KeyGen($\mathrm{MSK}, M=(Q,\mathcal{T},q_0,F)$*)* The key generation algorithm takes as input the master secret key and the description of a deterministic finite state machine M. The description of M includes a set Q of states $q_0,\ldots q_{|Q|-1}$ and a set of transitions $\mathcal{T}$ where each transition $t\in\mathcal{T}$ is a 3-tuple $(x,y,\sigma)\in Q\times Q\times\Sigma$. In addition, q_0 is designated as a unique start state and $F\subseteq Q$ is the set of accept states. (Recall, Σ is given in the parameters.)

The algorithm begins by choosing $|Q|$ random group elements $D_0, D_1, \ldots, D_{|Q|-1}\in\mathbb{G}$, where we associate D_i with state q_i. Next, for each $t\in\mathcal{T}$ it chooses random $r_t\in\mathbb{Z}_p$; it also chooses random $r_{\mathrm{start}}\in\mathbb{Z}_p$ and $\forall q_x\in F$ it chooses random $r_{\mathrm{end}_x}\in\mathbb{Z}_p$.

It begins creating the key with

$$K_{\mathrm{start}1} = D_0 (h_{\mathrm{start}})^{r_{\mathrm{start}}},\ K_{\mathrm{start}2} = g^{r_{\mathrm{start}}}$$

For each $t=(x,y,\sigma)\in\mathcal{T}$ the algorithm creates the key components

$$K_{t,1} = D_x^{-1} z^{r_t}, \quad K_{t,2} = g^{r_t}, \quad K_{t,3} = D_y (h_\sigma)^{r_t}.$$

Finally, for each $q_x\in F$ it computes

$$K_{\mathrm{end}x,1} = g^{-\alpha}\cdot D_x (h_{\mathrm{end}})^{r_{\mathrm{end}x}},\ K_{\mathrm{end}x,2} = g^{r_{\mathrm{end}x}}$$

$$\mathrm{SK} = \big(M, K_{\mathrm{start}1}, K_{\mathrm{start}2},\ \forall t\in\mathcal{T}\ (K_{t,1},K_{t,2},K_{t,3}),\ \forall q_x\in F\ (K_{\mathrm{end}x,1}, K_{\mathrm{end}x,2})\big)$$

Decrypt(SK, CT*)* Suppose we are given a ciphetext CT associated with a string $w = w_1, \ldots, w_\ell$ and a secret key SK associated with a DFA $M = (Q, \mathcal{T}, q_0, F)$ where $\textsc{Accept}(M, w)$. Then there exist a sequence of $\ell + 1$ states $u_0, u_1, \ldots, u_\ell$ and ℓ transitions $t_1, \ldots, t_\ell$ where $u_0 = q_0$ and $u_\ell \in F$ and for $i = 1, \ldots, \ell$ we have $t_i = (u_{i-1}, u_i, w_i) \in \mathcal{T}$.

The algorithm begins by computing:

$$B_0 = e(C_{\mathrm{start}1}, K_{\mathrm{start}1}) \cdot e(C_{\mathrm{start}2}, K_{\mathrm{start}2})^{-1} = e(g, D_0)^{s_0}.$$

Then for $i = 1$ to ℓ it iteratively computes:

$$B_i = B_{i-1} \cdot e(C_{(i-1),1}, K_{t_i,1}) e(C_{i,2}, K_{t_i,2})^{-1} e(C_{i,1}, K_{t_i,3}) = e(g, D_{u_i})^{s_i}$$

Since the machine accepts we have that $u_\ell = q_x$ for some $q_x \in F$ and $B_\ell = e(g, D_x)^{s_\ell}$.

It finally computes

$$B_{\mathrm{end}} = B_\ell \cdot e(C_{\mathrm{end}x,1}, K_{\mathrm{end}x,1})^{-1} \cdot e(C_{\mathrm{end}x,2}, K_{\mathrm{end}x,2}) = e(g, g)^{\alpha s_\ell}.$$

This value can then be divided from C_m to recover the message m.

3.2 Further Discussion

As discussed in the introduction, our construction contains three primary mechanisms used for decryption. The first step of the decryption process gives what in the introduction we called the initiation mechanism, which starts by computing $e(g, D_0)^{s_0}$. This used the "start" values from the keys and ciphertexts. We observe that this mechanism has structural similarities to the Boneh-Boyen [5] Identity-Based Encryption system.

The next several steps of decryption provide the transition mechanism. The evaluation of $e(C_{(i-1),1}, K_{t_i,1}) e(C_{i,2}, K_{t_i,2})^{-1} e(C_{i,1}, K_{t_i,3})$ results in the term $(e(g, D_x)^{s_{i-1}})^{-1} e(g, D_y)^{s_i}$, which updates the accumulated value to $e(g, D_y)^{s_i}$. Representing that the machine M is in state y after processing i symbols of w. A paramount feature is that the $C_{i,2}$ components of the ciphertext "chain" adjacent symbols together. This along with the structure of the private key enforces that $(e(g, D_x)^{s_{i-1}})^{-1} e(g, D_y)^{s_i}$ can only be computed if the i-th symbol is σ *and* the transition from state x to y on symbol σ is in the DFA for some σ.

Finally, the completion mechanism allows for the computation of $e(g, g)^{\alpha s_\ell}$ if the accumulated value reaches $e(g, D_x)^{s_\ell}$ for $q_x \in F$. The completion mechanism is very close in design to the initiation mechanism, but has the master key $g^{-\alpha}$ multiplied into its key component.

As described in the introduction, an attacker can backtrack to get some accumulated value that may not represent the actual computation of M on w. However, the attacker intuitively will only be able to backtrack so far and since M is deterministic must eventually go forward again to the same spot if he is to decrypt. This intuition is captured rigorously in the security proof in the next section.

3.3 Rerandomization of Ciphertexts and Secret Keys

We give two algorithms for the rerandomization of ciphertexts and private keys. Our ciphertext rerandomization algorithm takes in any well formed ciphertext for string w and produces a new ciphertext for the string w which encrypts the same message as the original. Moreover, the output ciphertext has the same distribution as one created fresh from the encryption algorithm. Similarly, the key rerandomization algorithm takes any valid secret key SK for DFA M and outputs a new secret key for DFA M that where the distribution is the same as the KeyGen algorithm.

These algorithms will be used for proving security of our system in the next section. Our main reduction techniques will produce valid challenge ciphertexts and private keys; however, these might not be well distributed. The randomization algorithms can then be applied to get the properly distribution on ciphertexts and keys. We segregate rerandomization as a separate step to help simplify the presentation of our proofs.[4] Our algorithms do simple additive (in the exponent) rerandomization. The presentation of these algorithms is in the full version of our paper.

4 Security Proof

We now prove security of our construction in the selective mode. We assume a successful attacker $\mathcal{A}$ against our system. Our reduction algorithm $\mathcal{B}$ will run $\mathcal{A}$ and use it to break the decision ℓ^*-Expanded BDHE assumption, where ℓ^* is the length of the string w^* used in creating the challenge ciphertext. Our reduction describes how $\mathcal{B}$ simulates the Setup, Key Generation, and Challenge Ciphertext generation phases of the security game.

We prove the following theorem.

Theorem 1. *Suppose the decision ℓ^*-Expanded BDHE assumption holds. Then no poly-time adversary can selectively break our DFA-based encryption system where the challenge string w^* encrypted is of length ℓ^*.*

Suppose we have an adversary $\mathcal{A}$ with non-negligible advantage $\epsilon = \mathsf{Adv}_{\mathcal{A}}$ in the selective security game against our construction for alphabet Σ. Moreover, suppose it chooses a challenge string w^* of length ℓ^*. Then $\mathcal{B}$ runs $\mathcal{A}$ and simulates the security game as follows.

Init $\mathcal{B}$ takes a decision ℓ^*-Expanded BDHE challenge $\boldsymbol{X}, T$. The attacker, $\mathcal{A}$, declares a challenge string w^* of length ℓ^*. We let ${w^*}_j$ denote the j-th symbol of w^*. In addition, we define ${w^*}_{\ell^*+1} = {w^*}_0 = \perp$ for a special symbol $\perp \notin \Sigma$.

[4] Most prior reductions of a similar nature (e.g. [5, 6, 28]) build the rerandomization directly in the main reduction.

Setup The reduction algorithm first chooses random exponents[5] $v_z, v_{\text{start}}, v_{\text{end}} \in \mathbb{Z}_p$ and $\forall \sigma \in \Sigma\ v_\sigma$. The parameter values are chosen as follows:

$$e(g,g)^\alpha = e(g^a, g^b), g = g, z = g^{v_z} g^{ab/d}$$

This implicitly sets $\alpha = ab$. Next it sets:

$$h_{\text{start}} = g^{v_{\text{start}}} \prod_{j \in [1,\ell^*]} g^{-a^j b/c_j}, \quad h_{\text{end}} = g^{v_{\text{end}}} \prod_{j \in [2,\ell^*+1]} g^{-a^j b/c_j}.$$

Finally,

$$\forall_{\sigma \in \Sigma}\ h_\sigma = g^{v_\sigma} g^{-b/d} \cdot \prod_{\substack{j \in [0,\ell^*+1] \text{ s.t.} \\ {w^*}_j \neq \sigma}} g^{-a^{(\ell^*+1-j)} b/c_{(\ell^*+1-j)}}.$$

The reduction algorithm embeds its knowledge of w^* into the the public parameters. The parameter h_σ will have a term $g^{-a^{\ell^*+1-j} b/c_{\ell^*+1-j}}$ if and only if the j-th symbol of w^* is not σ. As we will see this embedding will be crucial to simulating a challenge ciphertext, while maintaining the ability to generate keys. The terms are well distributed since a is chosen randomly in the assumption and due to the 'v' exponents, which randomize the other parameters. We also observe that this equation is well defined since we defined ${w^*}_{\ell^*+1} = {w^*}_0 = \perp$. Since $\perp \notin \Sigma$ we have that for all σ the parameter h_σ always contains the terms $g^{-ba^{(\ell^*+1)}/c_{(\ell^*+1)}}$ and g^{-ba^0/c_0}.

Challenge We describe how $\mathcal{B}$ creates a challenge ciphertext. The reduction will intuitively set $s_i = sa^i \in \mathbb{Z}_p$. $\mathcal{B}$ first chooses a coin β and begins to create a ciphertext CT. It first sets $w = w^*$ (from Init) and sets $C_m = m_\beta \cdot T$. Next, it sets

$$C_{\text{start}1} = g^s, \quad C_{\text{start}2} = (h_{\text{start}})^s = (g^s)^{v_{\text{start}}} \prod_{j \in [1,\ell^*]} g^{-a^j bs/c_j}$$

Then, for $i = 1$ to ℓ^*:

$$C_{i,1} = g^{a^i s}, \quad C_{i,2} = (g^{a^i s})^{v_{{w^*}_i}} (g^{a^{i-1} s})^{v_z} \cdot \prod_{\substack{j \in [0,\ell^*+1] \\ \text{s.t. } {w^*}_j \neq {w^*}_i}} g^{(-a^{\ell^*+1-j+i}) bs/c_j}.$$

We make two observations. First, the algorithm $\mathcal{B}$ does not receive from the assumption the term $g^{-a^{\ell^*+1} bs/c_j}$ for any j. Thus, it is important that such a term not be included in $C_{i,2}$. We see that such a term can only appear in the product above when $i = j$, however, this cannot be the case since if $i = j$ then ${w^*}_j = {w^*}_i$, which is explicitly excluded from the product.

[5] These values will be used to sure that are public parameters are distributed as in the real system. While this is of course necessary, they are not central to the core ideas of the reduction. As a simplification, a reader might choose to ignore these (imagine they are all 0) to help understand the main ideas.

Second, we remark that $z^{s_{i-1}}$ produces a term $g^{a^i bs/d}$ which $\mathcal{B}$ does not have. However, $(h_{w^*_i})^{s_i}$ produces a term which is the inverse of this and these cancel each other out in the computation of $C_{i,2} = z^{s_{i-1}}(h_{w^*_i})^{s_i}$. The remaining terms shown above are produceable from the assumption.

These two observations reflect important points in the proof. The first shows how the embedding of the challenge string w^* in the h_σ values allows us the creation of the challenge ciphertext. The second cancellation reflects the "linking" of adjacent symbols of w^* that is at the core of the security of the ciphertext construction.

Finally, it creates

$$C_{\text{end}\,1} = g^{a^{\ell^*} s}, \quad C_{\text{end}\,2} = (h_{\text{end}})^{a^{\ell^*} s} = (g^{a^{\ell^*} s})^{v_{\text{end}}} \prod_{j \in [2, \ell^*+1]} g^{-a^{\ell^*+j} bs/c_j}$$

To finish, the $\mathcal{B}$ must run the ciphertext rerandomization algorithm (specified in the full version) on CT to get a well distributed challenge ciphertext CT*. It then returns CT* to $\mathcal{A}$. If $T = g^{a^{\ell^*+1} bs}$ — is a valid Expanded BDHE tuple — then (the rerandomized) CT* is an encryption of m_β. Otherwise the ciphertext will reveal no information about β.

Key Generation The key generation phases (1 and 2) are the most complex part of this reduction. We describe how $\mathcal{B}$ can respond to a private key request for DFA $M = (Q, \mathcal{T}, q_0, F)$.

Before showing how $\mathcal{B}$ creates the actual key components we start by doing some prep work to show how the D_x values will implicitly be assigned. $\mathcal{B}$ will not actually be able to directly create these. We will describe the terms which comprise each D_x value and will create the keys to be consistent with these. Intuitively, the assignments should in some way match the execution of machine M on w^*.

We begin by introducing some notation. First we let $w^{*(i)}$ denote the *last* i symbols of w^*. It follows that $w^{*(\ell^*)} = w^*$ and $w^{*(0)}$ is the empty string. In addition, we for $k \in [0, |Q|-1]$ we let $M_k = (Q, \mathcal{T}, q_k, F)$. That is M_k is the same DFA as M except the start state is changed to q_k. (Note that $M_0 = M$.)

Now for each $q_k \in Q$ we create a set S_k of indices between 0 and ℓ^*. For $i = 0, 1, \ldots, \ell^*$ we put $i \in S_k$ if and only if $\textsc{Accept}(M_k, w^{*(i)})$. Then we assign

$$D_k = \prod_{i \in S_k} g^{a^{i+1} b}.$$

This assignment of D_k values is meant to "mark" the computation of M on the challenge string w^*. The term $g^{a^{i+1} b}$ will appear in D_k if it is possible to reach an accepting final state using the last i symbols of w^* starting at state q_k. We make two important observations for creating private keys. First, the term $g^{a^{\ell^*+1} b}$ *does not* appear in D_0. This follows from the fact that any valid request of a DFA M must reject w^* and is crucial to creating $K_{\text{start}1}$. Next for all $x \in F$, we have that the term $g^{a^1 b}$ *does* appear in D_x. This occurs since M_x will accept the empty string if it starts at an accepting state x and is a critical fact needed for creating $K_{\text{end}x,1}$.

The values embedded in D_k reflect more than just the computation of M on w^*. It embeds the execution of all M_k (M with all possible starting states) from all positions of the string w^*. This is done to reflect the possible backtracking attacks described in the last section. We emphasize that $\mathcal{B}$ cannot actually produce these D_k values using the terms given from the assumption. Instead it will construct the key components to be consistent with these values. Uncomputable terms will cancel when creating the components. $\mathcal{B}$ begins with creating the start and end key terms.

$\mathcal{B}$ starts by implicitly setting $r_{\text{start}} = \Sigma_{i \in S_0}\ c_{i+1}$.

$$K_{\text{start}2} = g^{r_{\text{start}}} = \prod_{i \in S_0} g^{c_{i+1}}$$

$$K_{\text{start}1} = D_0(h_{\text{start}})^{r_{\text{start}}} = (K_{\text{start}2})^{v_{\text{start}}} \cdot \prod_{\substack{j \in [1,\ell^*], i \in S_0 \\ j \neq i+1}} g^{-a^j b c_{i+1}/c_j}$$

Our assignments canceled out the terms of the form $g^{a^{i+1}b}$ from D_0 and the remaining terms that are given in $K_{\text{start}1}$ are those that we are given from the assumption. Notice that since the term $g^{a^{\ell^*+1}b}$ was not in D_0 it did not need to be canceled; this is important since the setting of h_{start} gave no way to do this. (The term $g^{a^{\ell^*+1}b/c_{\ell^*+1}}$ is not in h_{start}.)

Next, for all $q_x \in F$ it creates the key components. It first creates starts by implicitly setting $r_{\text{end}x} = \Sigma_{i \in S_x, i \neq 0}\ c_{i+1}$.

$$K_{\text{end}2,x} = g^{r_{\text{end}x}} = \prod_{\substack{i \in S_x \\ i \neq 0}} g^{c_{i+1}}$$

$$K_{\text{end}1,x} = g^{-\alpha} D_x(h_{\text{end}})^{r_{\text{end}x}} = (K_{\text{end}2,x})^{v_{\text{end}}} \cdot \prod_{\substack{j \in [2,\ell^*+1], i \in S_x \\ i \neq 0, j \neq i+1}} g^{-a^j b c_{i+1}/c_j}$$

Our assignment of $K_{\text{end}2,x}$ canceled out the terms of the form $g^{a^{i+1}b}$ from D_x except for when $i = 0$. Here, D_x has the term g^{ab} and this cancels with $g^{-\alpha} = g^{-ab}$. It is essential that D_x contain the term g^{ab} since h_{end} is structured such there is no other way of canceling with $g^{-\alpha}$. (The term $g^{a^1 b/c_1}$ is not in h_{end}.)

We finally need to create the key components for each $t = (x, y, \sigma) \in \mathcal{T}$. We organize this into a set of intermediate computations. For $i = 0$ to $\ell^* + 1$ we will define $(K_{t,1,i}, K_{t,2,i}, K_{t,3,i})$. Then we will let $K_{t,1} = \prod_{i \in [0,\ell^*+1]} K_{t,1,i},\ K_{t,2} = \prod_{i \in [0,\ell^*+1]} K_{t,2,i},\ K_{t,3} = \prod_{i \in [0,\ell^*+1]} K_{t,2,i}$.

Intuitively, for each transition $t = (x, y, \sigma)$ we step through each i from 0 to ℓ^*. [6] For each i we describe how to set $K_{t,2,i}$ such that it will cancel $g^{a^{i+1}}b$

[6] We explicitly note that -1 and $\ell^* + 1 \notin S_k$ for all k, which allows these cases to all be well defined for the range of i.

from D_x if this term appears in D_x and how to cancel $g^{a^i b}$ from D_y if this term appears in D_y in computing the $K_{t,1}, K_{t,3}$ key components. We step through four possible cases.

Case 1: $i \notin S_x \wedge (i-1) \notin S_y$

Set $K_{t,1,i}, K_{t,2,i}, K_{t,3,i} = 1$ (the identity element). This is when there are not term $g^{a^{i+1}b}$ in either D_x nor term $g^{a^i b}$ in D_y so nothing needs to be canceled.

Case 2: $i \in S_x \wedge (i-1) \in S_y$

Set $K_{t,2,i} = g^{a^i d}$ and $K_{t,1,i} = (K_{t,2,i})^{v_z}$.

$$K_{t,3,i} = (K_{t,2,i})^{v_\sigma} \cdot \prod_{\substack{j\in[0,\ell^*+1] \text{ s.t.} \\ {w^*}_j \neq \sigma}} g^{-a^{(\ell^*+1-j+i)}bd/c_{(\ell^*+1-j)}}$$

This is when there are terms $g^{a^{i+1}b}$ in D_x and $g^{a^i b}$ in D_y. The setting of $K_{t,2,i}$ allows them to both be canceled and the remaining terms above are "collateral" which can be taken from the assumption. This action is independent of the symbol ${w^*}_{i+1}$. We can think of the setting of $K_{t,2,i}$ as a "copy action" in that a similar cancellation happens on both sides of the transition.

Case 3: $i \notin S_x \wedge (i-1) \in S_y \wedge {w^*}_{\ell^*+1-i} \neq \sigma$

Set $K_{t,2,i} = g^{c_i}$ and $K_{t,1,i} = (K_{t,2,i})^{v_z} g^{abc_i/d}$.

$$K_{t,3,i} = (K_{t,2,i})^{v_\sigma} \cdot g^{-bc_i/d} \cdot \prod_{\substack{j\in[0,\ell^*+1] \text{ s.t.} \\ j\neq \ell^*+1-i \ \wedge \ {w^*}_j \neq \sigma}} g^{-a^{(\ell^*+1-j)}bc_i/c_{(\ell^*+1-j)}}$$

In this case there is *not* a term $g^{a^{i+1}b}$ in D_x, but there is a term $g^{a^i b}$ in D_y. Therefore we cannot apply the above "copy" technique from Case 2. Instead we use the fact that ${w^*}_{\ell^*+1-i} \neq \sigma$. (Note that ${w^*}_{\ell^*+1-i}$ is the first symbol of the string ${w^*}^{(i)}$ — the string of the last i symbols of w^*.) We set $K_{t,2,i} = g^{c_i}$ which cancels the only g^{a^i} term in D_y. Since ${w^*}_{\ell^*+1-i} \neq \sigma$ we have that h_σ contains the term $g^{-a^i b/c_i}$. Raising this to c_i provides the desired cancellation. The remaining terms shown above are "collateral" that can be taken from the assumption.

Case 4: $i \in S_x \wedge (i-1) \notin S_y \wedge {w^*}_{\ell^*+1-i} \neq \sigma$

First set, $K_{t,2,i} = g^{a^i d} g^{-c_i}$ and $K_{t,1,i} = (K_{t,2,i})^{v_z} g^{-abc_i/d}$.
Then, $K_{t,3,i} = (K_{t,2,i})^{v_\sigma} \cdot g^{bc_i/d}$.

$$\prod_{\substack{j\in[0,\ell^*+1] \text{ s.t.} \\ {w^*}_j \neq \sigma}} g^{-a^{(\ell^*+1-j+i)}bd/c_{(\ell^*+1-j)}} \prod_{\substack{j\in[0,\ell^*+1] \text{ s.t.} \\ j\neq \ell^*+1-i \ \wedge \ {w^*}_j \neq \sigma}} g^{a^{(\ell^*+1-j)}bc_i/c_{(\ell^*+1-j)}}$$

How we handle this case can best be understood as a combination of how we handle Cases 2 and three. Here there is a term $g^{a^{i+1}b}$ in D_x, but there is a *not* term $g^{a^i b}$ in D_y. We first put a term $g^{a^i d}$ in $K_{t,2,i}$ to invoke the "copy" mechanism from Case 2. This gives us the desired cancellation for part of D_x, but also creates an undesirable term $g^{a^i b}$ that cannot be cancelled with D_y. Therefore we also include a term of g^{-c_i} in in $K_{t,2,i}$ to invoke the cancelation of the "undesirable term". As in Case 3, we have that $w^*{}_{\ell^*+1-i} \neq \sigma$ and h_σ contains the term $g^{-a^i b/c_i}$. Again, remaining terms shown above are "collateral" which can be taken from the assumption. *We observe that these terms are basically just those generated from Case 2 and Case 3 combined.*

These four cases cover all possibilities. By the definition of a DFA when $w^*{}_{\ell^*+1-i} = \sigma$ we have that $i \in S_x$ if and only if $i-1 \in S_y$. This is a consequence of the fact that M is deterministic. (If M were instead a Nondeterministic Finite Automata, the prior statement would not hold.)

This shows how $\mathcal{B}$ creates all the key components. $\mathcal{B}$ must run the key rerandomization algorithm from the previous section on SK to get a well distributed key $\tilde{\text{SK}}$ for the machine M. Then it returns $\tilde{\text{SK}}$ to $\mathcal{A}$.

Guess The adversary will eventually output a guess β' of β. If $\beta = \beta'$, then $\mathcal{B}$ then outputs 0 to guess that $T = e(g,g)^{a^{\ell^*+1}bs}$; otherwise, it and outputs 1 to indicate that it believes T is a random group element in $\mathbb{G}_T$.

When T is a tuple the simulator $\mathcal{B}$ gives a perfect simulation so we have that

$$\Pr\left[\mathcal{B}\left(\boldsymbol{y}, T = e(g,g)^{a^{\ell^*+1}bs}\right) = 0\right] = \frac{1}{2} + \mathsf{Adv}_{\mathcal{A}}.$$

When T is a random group element the message $\mathcal{M}_\beta$ is completely hidden from the adversary and we have $\Pr\left[\mathcal{B}\left(\boldsymbol{X}, T = R\right) = 0\right] = \frac{1}{2}$. Therefore, $\mathcal{B}$ can play the decision ℓ^*-Expanded BDHE game with non-negligible advantage.

5 Conclusions

We introduced a new type of functional encryption system that works over regular languages. Our system has secret keys that encode a Deterministic Finite Automata M and ciphertexts that are associated with an arbitrary length string w and an encrypted message m. A user with a secret key for DFA M can decrypt a ciphertext associated with w if and only if M accepts w. Our construction makes use of bilinear maps and constructs mechanisms that enforce the DFA evaluation.

Interesting future challenges include developing a system that can be proved adaptively secure and under a non-parameterized assumption. Looking farther out, one would like to be able to extend the functionality further, eventually all the way to support Turing Machines.

Acknowledgements

We thank Dan Boneh, Allison Lewko, and Amit Sahai for helpful comments and feedback.

References

1. 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.
2. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (h)ibe in the standard model. In *EUROCRYPT*, pages 553–572, 2010.
3. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical ibe. In *CRYPTO*, pages 98–115, 2010.
4. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.
5. Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.
6. Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.
7. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.
8. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.
9. 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.
10. Dan Boneh and Mike Hamburg. Generalized identity-based and broadcast encryption schemes. In *Proc. of Asiacrypt*, pages 455–470, 2008.
11. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.
12. Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.
13. David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552, 2010.
14. Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.
15. Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.
16. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.
17. Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In *ICALP (2)*, pages 579–591, 2008.
18. 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.
19. Mike Hamburg. Spatial encryption. *IACR Cryptology ePrint Archive*, 2011:389, 2011.
20. Vincenzo Iovino and Giuseppe Persiano. Hidden-vector encryption with groups of prime order. In *Pairing*, pages 75–88, 2008.

21. Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.
22. 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.
23. Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, pages 214–231, 2009.
24. Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.
25. Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.
26. Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.
27. M. Sipser. *Introduction to the theory of computation*. Thomson Course Technology, 2006.
28. Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.
29. Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636, 2009.
30. Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. PKC, 2011.