Attribute-Based Encryption for Circuits from Multilinear Maps

Sanjam Garg Craig Gentry Shai Halevi Amit Sahai Brent Waters

Abstract

In this work, we provide the first construction of Attribute-Based Encryption (ABE) for general circuits. Our construction is based on the existence of multilinear maps. We prove selective security of our scheme in the standard model under the natural multilinear generalization of the BDDH assumption. Our scheme achieves both Key-Policy and Ciphertext-Policy variants of ABE.

Our scheme and its proof of security directly translate to the recent multilinear map framework of Garg, Gentry, and Halevi.

This paper subsumes the manuscript of Sahai and Waters [SW12].

1 Introduction

In traditional public key encryption a sender will encrypt a message to a targeted individual recipient using the recipient's public key. However, in many applications one may want to have a more general way of expressing who should be able to view encrypted data. Sahai and Waters [SW05] introduced the notion of Attribute-Based Encryption (ABE). There are two variants of ABE: Key-Policy ABE and Ciphertext-Policy ABE [GPSW06]. (We will consider both these variants in this work.) In a Key-Policy ABE system, a ciphertext encrypting a message M is associated with an assignment x of boolean variables. A secret key SK is issued by an authority and is associated with a boolean function f chosen from some class of allowable functions $\mathcal{F}$. A user with a secret key for f can decrypt a ciphertext associated with x, if and only if $f(x) = 1$.

Since the introduction of ABE there have been advances in multiple directions. These include new proof techniques to achieve adaptive security [LOS+10, OT10, LW12], decentralizing trust among multiple authorities [Cha07, CC09, LW11], and applications to outsourcing computation [PRV12].

However, the central challenge of expanding the *class* of allowable boolean functions $\mathcal{F}$ has been very resistant to attack. Viewed in terms of circuit classes, the work of Goyal *et al* [GPSW06] achieved the best result until now: their construction achieved security essentially for circuits in the complexity class $\mathbf{NC^1}$. This is the class of circuits with depth $\log n$, or equivalently, the class of functions representable by polynomial-size boolean formulas. Achieving ABE for general circuits is arguably the central open direction in this area[1].

Difficulties in achieving Circuit ABE and the Backtracking Attack. To understand why achieving ABE for general circuits has remained a difficult problem, it is instructive to examine the mechanisms of existing constructions based on bilinear maps. Intuitively, a bilinear map allows

[1] We note that if collusions between secret key holders are bounded by a publicly known polynomially-bounded number in advance, then even stronger results are known [SS10, GVW12]. However, throughout this paper we will deal only with the original setting of ABE where unbounded collusions are allowed between adversarial users.

one to decrypt using groups elements as keys (or key components) as opposed to exponents. By handing out a secret key that consists of group elements an authority is able to computationally hide some secrets embedded in that key from the key holder herself. In contrast, if a secret key consists of exponents in $\mathbb{Z}_p$ for a prime order group p, as in say an ElGamal type system, then the key holder or collusion of key holders can solve for these secrets using algebra. This computational hiding in bilinear map based systems allows an authority to personalize keys to a user and prevent collusion attacks, which are the central threat.

Using GPSW [GPSW06] as a canonical example we illustrate some of the main principles of decryption. In their system, private keys consists of bilinear group elements for a group of prime order p and are associated with random values $r_y \in \mathbb{Z}_p$ for each leaf node in the boolean formula f. A ciphertext encrypted to descriptor x has randomness $s \in \mathbb{Z}_p$. The decryption algorithm begins by applying a pairing operation to each "satisfied" leaf node and obtains $e(g,g)^{r_y s}$ for each satisfied node y. From this point onward decryption consists solely of finding if there is a linear combination (in the exponent) of the r_y values that can lead to computing $e(g,g)^{\alpha s}$ which will be the "blinding factor" hiding the message M. (The variable $e(g,g)^{\alpha}$ is defined in the public parameters.) The decryption algorithm should be able to find such a linear combination only if $f(x) = 1$. Of particular note is that once the $e(g,g)^{r_y s}$ values are computed the pairing operation plays no further role in decryption. Indeed it cannot, since it is intuitively "used up" on the initial step.

Let's now take a closer look at how GPSW structures the private keys given a boolean formula. Suppose in a boolean formula that there consisted an OR gate T that received inputs from gates A and B. Then the authority would associate gate T with a value r_T and gates A, B with values $r_A = r_B = r_T$ to match the OR functionality. Now suppose that on a certain input assignment x that gate A evaluates to 1, but gate B evaluates to 0. The decryptor will then learn the "decryption value" $e(g,g)^{s r_A}$ for gate A and can interpolate up by simply by noting that $e(g,g)^{s r_T} = e(g,g)^{s r_A}$. While this structure reflects an OR gate, it also has a critical side effect. The decryption algorithm also learns the decryption value $e(g,g)^{s r_B}$ for gate B *even though gate B evaluates to 0* on input x. We call such a discovery a *backtracking attack*.

Note that boolean formulas are circuits with fanout one. If the fanout is one, then the backtracking attack produces no ill effect since an attacker has nowhere else to go with this information that he has learned. However, suppose we wanted to extend this structure with circuits of fanout of two or more, and that gate B also fed into an AND gate R. In this case the backtracking attack would allow an attacker to act like B was satisfied in the formula even though it was not. This misrepresentation can then be propagated up a different path in the circuit due to the larger fanout. (Interestingly, this form of attack does not involve collusion with a second user.)

We believe that such backtracking attacks are the principle reason that the functionality of existing ABE systems has been limited to circuits of fanout one. Furthermore, we conjecture that since the pairing operation is used up in the initial step, that there is no black-box way of realizing general ABE for circuits from bilienar maps.

Our Results. We present a new methodology for constructing Attribute-Based Encryption systems for circuits of arbitrary fanout. Our method is described using multilinear maps. Cryptography with multilinear maps was first postulated by Boneh and Silverberg where they discussed potential applications such as one round, n-way Diffie-Hellman key exchange. However, they also gave evidence that it might be difficult or not possible to find useful multilinear forms within the realm of algebraic geometry. For this reason there has existed a general reluctance among cryptographers to explore multilinear map constructions even though in some constructions such

as the Boneh-Goh-Nissim [BGN05] slightly homomorphic encryption system, or the Boneh-Sahai-Waters [BSW06] Traitor Tracing scheme, there appears to exist direct generalizations of bilinear map solutions.

Very recently, Garg, Gentry, and Halvei [GGH12] announced a surprising result. Using ideal lattices they produced a candidate mechanism that would approximate or be the moral equivalent of multilinear maps for many applications. Speculative applications include translations of existing bilinear map constructions and direct generalizations as well as future applications. While the development and cryptanalysis of their tools is at a nascent stage, we believe that their result opens an exciting opportunity to study new constructions using a multilinear map abstraction. The promise of these results is that such constructions can be brought over to their framework or a related future one. We believe that building ABE for circuits is one of the most exciting of these problems due to the challenges discussed above and that existing bilinear map constructions do not have a direct generalization.

Our circuit ABE construction and its proof of security directly translate to the framework of [GGH12].

We construct an ABE system of the Key-Policy variety where ciphertext descriptors are an n-tuple x of boolean variables and keys are associated with boolean circuits of a max depth ℓ, where both ℓ and n are polynomially bounded and determined at the time of system setup. Our main construction exposition is for circuits that are layered (where gates at depth j get inputs from gates at depth $j-1$) and monotonic (consisting only of AND plus OR gates). Neither one of these impacts are general result as a generic circuit can be transformed into a layered one for the same function with a small amount of overhead. In addition, using DeMorgan's law one can build a general circuit from a monotone circuit with negation only appearing at the input wires. We sketch this in Section 2. We finally note that using universal circuits we can realize "Ciphertext-Policy" style ABE systems for circuits.

Our framework of multi-linear maps is that a party can call a group generator $\mathcal{G}(1^\lambda, k)$ to obtain a sequence of groups $\vec{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ each of large prime[2] order $p > 2^\lambda$ where each comes with a canonical generator $g = g_1, \ldots, g_k$. Slightly abusing notation, if $i+j \leq k$ we can compute a bilinear map operation on $g_i^a \in \mathbb{G}_i, g_j^b \in \mathbb{G}_j$ as $e(g_i^a, g_j^b) = g_{i+j}^{ab}$. These maps can be seen as implementing multilinear maps[3]. It is the need to commit to a certain k value which will require the setup algorithm of our construction to commit to a maximum depth $\ell = k-1$. We will prove security under a generalization of the decision BDH assumption that we call the decision k-multilinear assumption. Roughly, it states that given $g, g^s, g^{c_1}, \ldots, g^{c_k}$ it is hard to distinguish $T = g_k^{s \prod_{j \in [1,k]} c_k}$ from a random element of $\mathbb{G}_k$.

Our Techniques. As discussed there is no apparent generalization of the GPSW methods for achieving ABE for general circuits. We develop new techniques with a focus on preventing the backtracking attacks we described above. Intuitively, we describe our techniques as "move forward and shift"; this *replaces and subsumes* the linear interpolation method of GPSW decryption. In particular, our schemes do not rely on any sophisticated linear secret sharing schemes, as was done by GPSW.

Consider a private key for a given monotonic[4] circuit f with max depth ℓ that works over a

[2]We stress that our techniques do not rely on the groups being of prime order; we only need that certain randomization properties hold in a statistical sense (which hold perfectly over groups of prime order). Therefore, our techniques generalize to other algebraic settings.

[3]We technically consider the existence of a set of bilinear maps $\{e_{i,j} : G_i \times G_j \to G_{i+j} \mid i,j \geq 1;\ i+j \leq k\}$, but will often abuse notation for ease of exposition.

[4]Recall that assuming that the circuit is monotonic is without loss of generality. Our method also applies to

group sequence $(\mathbb{G}_1, \ldots, \mathbb{G}_k)$. Each wire w in f is associated by the authority with a random value $r_w \in \mathbb{Z}_p$. A ciphertext for descriptor x will be associated with randomness $s \in \mathbb{Z}_p$. A user should with secret key for f should be able to decrypt if and only if $f(x) = 1$.

The decryption algorithm works by computing $g_{j+1}^{sr_w}$ for each wire w in the circuit that evaluates to 1 on input x. If the wire is 0, the decryptor should not be able to obtain this value. Decryption works from the bottom up. For each input wire w at depth 1, we compute $g_2^{sr_w}$ using a very similar mechanism to GPSW.

We now turn our attention to OR gates to illustrate how we prevent backtracking attacks. Suppose wire w is the output of an OR gate with input wires $A(w), B(w)$ at depth j. Furthermore, suppose on a given input x the wire $A(w)$ evaluates to true and $B(w)$ to false so that the decryptor has $g_j^{sr_{A(w)}}$, but not $g_j^{sr_{B(w)}}$. The private key components associated with wire w are:

$$g^{a_w},\ g^{b_w},\ g_j^{r_w - a_w \cdot r_{A(w)}},\ g_j^{r_w - b_w \cdot r_{B(w)}}$$

for random a_w, b_w. To move decryption onward the algorithm first computes

$$e\left(g^{a_w}, g_j^{sr_{A(w)}}\right) = g_{j+1}^{sa_w r_{A(w)}}$$

This is the move forward step. Then it computes

$$e\left(g^{s}, g_j^{r_w - a_w \cdot r_{A(w)}}\right) = g_{j+1}^{s(r_w - a_w r_{A(w)})}$$

This is the shift step. Multiplying these together gives the desired term $g_{j+1}^{sr_w}$.

Let's examine backtracking attacks in this context. Recall that the attacker's goal would be to compute $g_j^{sr_{B(w)}}$ even though wire $B(w)$ is 0, and propagate this forward. From the output term and the fourth key component the attacker can actually inverse the shift process on the B side and obtain $g_{j+1}^{sa_w r_{A(w)}}$, however, since the map e works only in the "forward" direction, it is not possible to invert the move forward step and complete the attack. The crux of our security lies in this idea. In the main body of this paper we give our formal proof that captures this intuition.

The AND gate mechanism has a similar shift and move forward structure, but requires both inputs for decryption. If this process is applied iteratively, to an output gate $\tilde{w}$ then one obtains $g_k^{sr_{\tilde{w}}}$. A final header portion of the key and decryption mechanism is used to obtain the message. This portion is similar to prior work.

1.1 Other Related Work

Other recent functionality in a similar vain to ABE includes spatial encryption [Ham11] and regular language functionality [Wat12]. Neither of these seem to point to a path for achieving the general case of circuits. Indeed, [Wat12] argues that backtracking attacks as the reason that the constructions can only support Deterministic Finitie Automata and not Nondeterministic Finite Automata.

An interesting challenge going forward is whether new techniques can be applied to the general case of functional encryption [SW08, BSW11]. In this setting we would like to hide the input x as well as the message. So far the strongest functionality in this setting has been the inner product functionality of Katz, Sahai, and Waters [KSW08] and different variants of this [OT12].

There have been different lattice based constructions of IBE, HIBE, and Fuzzy IBE [CHKP10, ABB10, ABV$^+$12]. While the high level proof structures of these systems follow the earlier bilinear

general circuits that involve negations. See Section 2.

map counterparts closely, the analogies seem to break down at lower level mechanisms. For example, there is more asymmetry in the construction of keys and ciphertexts — in bilinear maps they were both bilinear group elements. Rothblum [Rot12] considers the problem of circular security from bit encryption systems from ℓ-multilinear maps. He considers a different form than us where ℓ group elements of different types are input at once to a multilinear map function. The assumption used is a variant of XDH.

Parno, Raykova and Vaikuntanathan [PRV12] note that delegation from ABE can be achieved from a system that is not collusion resistant, however, they were not able to leverage this to go beyond the boolean formulas of [GPSW06]. The fact that the backtracking attacks described above do not use collusion attacks, but are attacks within a key might help explain this. It is not clear if our techniques will be of immediate use to this type of delegation as the size of group elements and computation of group operations could grow with the sequence number k and thus the depth of the circuit. For a similar reason it is not clear if our techniques can be used to improve [Wat12].

Concurrent Work. Concurrent to and independent of our work Gorbunov, Vaikuntanathan, and Wee [GVW13] achieve ABE for circuits. One nice feature of their result is that they reduce security to the Learning with Errors (LWE) problem. Both our result and theirs has "succinct" ciphertexts in that the ciphertext size grows with the maximum depth of the circuits and not the size. Goldwasser, Kalai, Popa, Vaikuntanathan, and Zeldovich [GKP+13] show how one use such an ABE plus fully homomorphic encryption into a succinct single use functional encryption scheme. This in turn implies results for reusable Yao garbled circuits and other applications.

Subsequent Work. Subsequent to our work Garg, Gentry, Sahai, and Waters [GGSW13] showed that a general primitive they termed witness encryption implies circuit ABE if we have witness indistinguishable proofs. Their techniques of moving from witness encryption to ABE are quite different from our direct construction. A drawback of using witness encryption is that current GGSW constructions rely on a different assumption for each NP instance. In addition, the schemes are significantly much less practical due to the reduction to the Exact Cover problem.

1.2 Roadmap

We start by providing preliminary definition in Section 2. We give our construction based on multilinear maps and its proof os security in Section 3 and Section 4 respectively. We provide the translation of the scheme and the proof to the GGH framework [GGH12] in Appendix B and Appendix C respectively.

2 Preliminaries

In this section, we provide some preliminaries. These include discussion of monotone versus general circuits, our multi-linear map convention and assumptions, and our circuit notation. We place our security definition for ABE for circuits in Appendix A as it essentially follows [GPSW06] with the exception that access structures are circuits.

2.1 General Circuits vs. Monotone Circuits

We begin by observing that there is a folklore transformation that uses De Morgan's rule to transform any general Boolean circuit into an equivalent monotone Boolean circuit, with negation gates only allowed at the inputs. For completeness, we sketch the construction here.

Given a Boolean circuit C, consider the Boolean circuit $\tilde{C}$ that computes the negation of C. Note that such a circuit can be generated by simply recursively applying De Morgan's rule to each gate of C starting at the output gate. The crucial property of this transformation is that in this circuit $\tilde{C}$ each wire computes the negation of the corresponding original wire in C.

Now, we can construct a monotone circuit M by combining C and $\tilde{C}$ as follows: take each negation gate inside C, eliminate it, and replace the output of the negation gate by the corresponding wire in $\tilde{C}$. Do the same for negation gates in $\tilde{C}$, using the wires from C. In the end, this will yield a monotone circuit M with negation gates remaining only at the input level, as desired. The size of M will be no more than twice the original size of C, and the depth of M will be identical to the depth of C, where depth is computed ignoring negation gates. The correctness of this transformation follows trivially from De Morgan's rule.

As a result, we can focus our attention on monotone circuits. Note that inputs to the circuit correspond to boolean variables x_i, and we can simply introduce explicit separate attributes corresponding to $x_i = 0$ and $x_i = 1$. Honest encryptors are instructed to only set one of these two attributes for each variable x_i.

Because of this simple transformation, in the sequel, we will only consider ABE for monotone circuits.

2.2 Multi-linear maps

We assume the existence of a group generator $\mathcal{G}$, which takes as input a security parameter n and a positive integer k to indicate the number of allowed pairing operations. $\mathcal{G}(1^\lambda, k)$ outputs a sequence of groups $\vec{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ each of large prime order $p > 2^\lambda$. In addition, we let g_i be a canonical generator of $\mathbb{G}_i$ (and is known from the group's description). We let $g = g_1$.

We assume the existence of a set of bilinear maps $\{e_{i,j} : G_i \times G_j \to G_{i+j} \mid i, j \geq 1;\ i + j \leq k\}$. The map $e_{i,j}$ satisfies the following relation:

$$e_{i,j}\left(g_i^a, g_j^b\right) = g_{i+j}^{ab} \ : \ \forall a, b \in \mathbb{Z}_p$$

We observe that one consequence of this is that $e_{i,j}(g_i, g_j) = g_{i+j}$ for each valid i, j.

When the context is obvious, we will sometimes abuse notation drop the subscripts i, j, For example, we may simply write:

$$e\left(g_i^a, g_j^b\right) = g_{i+j}^{ab}$$

We define the k-Multilinear Decisional Diffie-Hellman (k-MDDH) assumption as follows:

Assumption 2.1 (k-Multilinear Decisional Diffie-Hellman: k-MDDH)**.** *The k-Multilinear Decisional Diffie-Hellman (k-MDDH) problem states the following: A challenger runs $\mathcal{G}(1^\lambda, k)$ to generate groups and generators of order p. Then it picks random $s, c_1, \ldots, c_k \in \mathbb{Z}_p$.*

The assumption then states that given $g = g_1, g^s, g^{c_1}, \ldots, g^{c_k}$ it is hard to distinguish $T = g_k^{s \prod_{j \in [1,k]} c_j}$ from a random group element in $\mathbb{G}_k$, with better than negligible advantage (in security parameter λ).

2.3 Circuit Notation

We now define our notation for circuits that adapts the model and notation of Bellare, Hoang, and Rogaway [BHR12] (Section 2.3). For our application we restrict our consideration to certain classes of boolean circuits. First, our circuits will have a single output gate. Next, we will consider layered

circuits. In a layered circuit a gate at depth j will receive both of its inputs from wires at depth $j-1$. Finally, we will restrict ourselves to monotonic circuits where gates are either AND or OR gates of two inputs. [5]

Our circuits will be a five tuple $f = (n, q, A, B, \texttt{GateType})$. We let n be the number of inputs and q be the number of gates. We define inputs $= \{1, \ldots, n\}$, Wires $= \{1, \ldots, n+q\}$, and Gates $= \{n+1, \ldots, n+q\}$. The wire $n+q$ is the designated output wire. A : Gates $\rightarrow$ Wires/outputwire is a function where $A(w)$ identifies w's first incoming wire and B : Gates $\rightarrow$ Wires/outputwire is a function where $B(w)$ identifies w's second incoming wire. Finally, $\texttt{GateType}$: Gates $\rightarrow \{\text{AND}, \text{OR}\}$ is a function that identifies a gate as either an AND or OR gate.

We require that $w > B(w) > A(w)$. We also define a function $\texttt{depth}(w)$ where if $w \in$ inputs $\texttt{depth}(w) = 1$ and in general $\texttt{depth}(w)$ of wire w is equal to the shortest path to an input wire plus 1. Since our circuit is layered we require that for all $w \in$ Gates that if $\texttt{depth}(w) = j$ then $\texttt{depth}(A(w)) = \texttt{depth}(B(w)) = j - 1$.

We will abuse notation and let $f(x)$ be the evaluation of the circuit f on input $x \in \{0,1\}^n$. In addition, we let $f_w(x)$ be the value of wire w of the circuit on input x.

3 Our Construction: Multilinear maps

We now describe our construction. Our main construction is of the Key-Policy form where a key generation algorithm takes in the description of a circuit f and encryption takes in an input x and message M. A user with secret key for f can decrypt if and only if $f(x) = 1$. The system is of the "public index" variety in that only the message M is hidden while x can be efficiently discovered from the ciphertext, as is standard for ABE. We will also discuss how our KP-ABE scheme yields a Ciphertext-Policy ABE scheme for bounded-size circuits.

The setup algorithm will take as inputs a maximum depth ℓ of all the circuits as well as the input size n for all ciphertexts. All circuits f in our system will be of depth ℓ (have the output gate at depth ℓ) and be layered as discussed in Section 2.3. Using layered circuits and having all circuits be of the same depth is primarily for ease of exposition, as we believe that our construction could directly be adapted to the general case. The fact that setup defines a maximum depth ℓ is more fundamental as the algorithm defines a $k = \ell + 1$ group sequence a k pairings.

Setup$(1^\lambda, n, \ell)$. The setup algorithm takes as input, a security parameter λ, the maximum depth ℓ of a circuit, and the number of boolean inputs n.

It then runs $\mathcal{G}(1^\lambda, k = \ell + 1)$ and of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ of prime order p, with canonical generators $g_1, \ldots, g_k$. We let $g = g_1$. Next, it chooses random $\alpha \in \mathbb{Z}_p$ and $h_1, \ldots, h_n \in \mathbb{G}_1$.

The public parameters, PP, consist of the group sequence description plus:

$$g_k^\alpha, h_1, \ldots, h_n$$

The master secret key MSK is $(g_{k-1})^\alpha$.

Encrypt(PP$, x \in \{0,1\}^n, M \in \{0,1\}$). The encryption algorithm takes in the public parameters, an descriptor input $x \in \{0,1\}^n$, and a message bit $M \in \{0,1\}$.

The encryption algorithm chooses a random s. If $M = 0$ it sets C_M to be a random group element in $\mathbb{G}_k$; otherwise it lets $C_M = (g_k^\alpha)^s$. Next, let S be the set such of i such that $x_i = 1$.

[5] These restrictions are mostly useful for exposition and do not impact functionality. General circuits can be built from non-monotonic circuits. In addition, given a circuit an equivalent layered exists that is larger by at most a polynomial factor.

The ciphertext is created as

$$\mathrm{CT} = (C_M,\ g^s,\ \forall i \in S\ \ C_i = h_i^s)$$

KeyGen(MSK, $f = (n, q, A, B, \texttt{GateType})$**).** The algorithm takes in the master secret key and a description f of a circuit. Recall, that the circuit has $n + q$ wires with n input wires, q gates and the wire $n + q$ designated as the output wire.

The key generation algorithm chooses random $r_1, \ldots, r_{n+q} \in \mathbb{Z}_p$, where we think of randomness r_w as being associated with wire w. The algorithm produces a "header" component

$$K_H = (g_{k-1})^{\alpha - r_{n+q}}$$

Next, the algorithm generates key components for every wire w. The structure of the key components depends upon if w is an input wire, an OR gate, or an AND gate. We describe how it generates components for each case.

- *Input wire*
  By our convention if $w \in [1, n]$ then it corresponds to the w-th input. The key generation algorithm chooses random $z_w \in \mathbb{Z}_p$.

  The key components are:
  $$K_{w,1} = g^{r_w} h_w^{z_w},\ K_{w,2} = g^{-z_w}$$

- *OR gate*
  Suppose that wire $w \in$ Gates and that $\texttt{GateType}(w) = \mathrm{OR}$. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w. The algorithm will choose random $a_w, b_w \in \mathbb{Z}_p$. Then the algorithm creates key components:

  $$K_{w,1} = g^{a_w},\ K_{w,2} = g^{b_w},\ K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)}},\ K_{w,4} = g_j^{r_w - b_w \cdot r_{B(w)}}$$

- *AND gate*
  Suppose that wire $w \in$ Gates and that $\texttt{GateType}(w) = \mathrm{AND}$. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w. The algorithm will choose random $a_w, b_w \in \mathbb{Z}_p$.

  $$K_{w,1} = g^{a_w},\ K_{w,2} = g^{b_w},\ K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

We will sometimes refer to the $K_{w,3}, K_{w,4}$ of the AND and OR gates as the "shift" components. This terminology will take on more meaning when we see how they are used during decryption.

The secret key SK output consists of the description of f, the header component K_H and the key components for each wire w.

Decrypt(SK, CT**).** Suppose that we are evaluating decryption for a secret key associated with a circuit $f = (n, q, A, B, \texttt{GateType})$ and a cipherext with input x. We will be able to decrypt if $f(x) = 1$.

We begin by observing that the goal of decryption should be to compute $g_k^{\alpha s}$ such that we can test if this is equal to C_M. First, there is a header computation where we compute $E' = e(K_H), g^s) = e(g_{k-1}^{\alpha - r_{n+q}}, g^s) = g_k^{\alpha s} g_k^{-r_{n+q} \cdot s}$ Our goal is now reduced to computing $g_k^{r_{n+q} \cdot s}$.

Next, we will evaluate the circuit from the bottom up. Consider wire w at depth j; if $f_w(x) = 1$ then, our algorithm will compute $E_w = (g_{j+1})^{s r_w}$. (If $f_w(x) = 0$ nothing needs to be computed for

that wire.) Our decryption algorithm proceeds iteratively starting with computing E_1 and proceeds in order to finally compute E_{n+q}. Computing these values in order ensures that the computation on a depth $j-1$ wire (that evaluates to 1) will be defined before computing for a depth j wire. We show how to compute E_w for all w where $f_w(x) = 1$, again breaking the cases according to whether the wire is an input, AND or OR gate.

- *Input wire*
  By our convention if $w \in [1, n]$ then it corresponds to the w-th input. Suppose that $x_w = f_w(x) = 1$. The algorithm computes:

  $$E_w = e(K_{w,1}, g^s) \cdot e(K_{w,2}, C_w) = e(g^{r_w} h_w^{z_w}, g^s) \cdot e(g^{-z_w}, h_w^s) = g_2^{sr_w}$$

  We observe that this mechanism is similar to many existing ABE schemes.

- *OR gate*
  Consider a wire $w \in$ Gates and that $\texttt{GateType}(w) = \text{OR}$. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w. Suppose that $f_w(x) = 1$. If $f_{A(w)}(x) = 1$ (the first input evaluated to 1) then we compute:

  $$E_w = e(E_{A(w)}, K_{w,1}) \cdot e(K_{w,3}, g^s) = e(g_j^{sr_{A(w)}}, g^{a_w}) \cdot e(g_j^{r_w - a_w \cdot r_{A(w)}}, g^s) = (g_{j+1})^{sr_w}$$

  Alternatively, if $f_{A(w)}(x) = 0$, but $f_{B(w)}(x) = 1$, then we compute:

  $$E_w = e(E_{B(w)}, K_{w,2}) \cdot e(K_{w,4}, g^s) = e(g_j^{sr_{B(w)}}, g^{b_w}) \cdot e(g_j^{r_w - b_w \cdot r_{B(w)}}, g^s) = (g_{j+1})^{sr_w}$$

  Let's exam this mechanism for the case where the first input is 1 ($f_{A(w)}(x) = 1$). In this case the algorithm "moves" the value $E_{A(w)}$ from group $\mathbb{G}_j$ to group $\mathbb{G}_{j+1}$ when pairing it with $K_{w,1}$. It then multiplies it by $e(K_{w,3}, g^s)$ which "shifts" that result to E_w.

  Suppose that $f_{A(w)}(x) = 1$, but $f_{B(w)}(x) = 0$. A critical feature of the mechanism is that an attacker cannot perform a "backtracking" attack to compute $E_{B(w)}$. The reason is that the pairing operation cannot be reverse to go from group $\mathbb{G}_{j+1}$ to group $\mathbb{G}_j$. If this were not the case, it would be debilitating for security as gate $B(w)$ might have fanout greater than 1. This type of backtracking attacking is why existing ABE constructions are limited to circuits with fanout of 1.

- *AND gate*
  Consider a wire $w \in$ Gates and that $\texttt{GateType}(w) = \text{AND}$. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w. Suppose that $f_w(x) = 1$. Then $f_{A(w)}(x) = f_{B(w)}(x) = 1$ and we compute:

  $$E_w = e(E_{A(w)}, K_{w,1}) \cdot e(E_{B(w)}, K_{w,2}) \cdot e(K_{w,3}, g^s)$$
  $$= e(g_j^{sr_{A(w)}}, g^{a_w}) \cdot e(g_j^{sr_{B(w)}}, g^{b_w}) \cdot e(g_j^{r_w - a_w \cdot r_{A(w)} - c_w \cdot r_{B(w)}}, g^s) = (g_{j+1})^{sr_w}$$

If the $f(x) = f_{n+q}(x) = 1$, then the algorithm will compute $E_{n+q} = g_k^{r_{n+q} \cdot s}$. It finally computes $E' \cdot E_{n+q} = g_k^{\alpha s}$ and tests if this equals C_M, outputting $M = 1$ if so and $M = 0$ otherwise. Correctness holds with high probability.

A Few Remarks. We end this section with a few remarks. First, the encryption algorithm takes as input a single bit message. In this setting we could imagine encoding a longer message by XORing it with the hash of $g_k^{\alpha s}$. However, we used bit encryption with a testability function so that our construction relies as minimally as possible on the exact algebraic representation of the multilinear map.

Our OR and AND key components respectively have one and two "shift" components. It is conceivable to have a construction with one shift component for the OR and none for the AND. However, we designed it this way since it made the exposition of our proof (in particular the distribution of private keys) easier.

Finally, our construction uses a layered circuit, where a wire at depth j gets its inputs from depth $j' = j - 1$. We could imagine a small modification to our construction which allowed j' to be of any depth less than j. Suppose this were the case for the first input. Then instead of $K_{w,1} = g_1^{a_w}$ we might more generally let $K_{w,1} = (g_{j-j'})^{a_w}$. However, we stick to describing and proving the layered case for simplicity.

4 Proof of Security

We prove (selective) security in the security model given by GPSW [GPSW06], where the key access structures are monotonic circuits. For a circuit of max depth $k-1$ we prove security under the decision k-multilinear assumption.

We show that if there exist a poly-time attacker $\mathcal{A}$ on our ABE system for circuits of depth ℓ and inputs of length n in the selective security game then we can construct a poly-time algorithm on the decision $\ell + 1$-multilinear assumption with non-neglgibile advantage. We describe how $\mathcal{B}$ interacts with $\mathcal{A}$.

Theorem 4.1. *The construction given in the previous section achieves selective security for arbitrary circuits of depth $k-1$ in the KP-ABE security game under the k-MDDH assumption.*

By using universal circuits, we obtain as an immediate corollary:

Corollary 4.2. *There exists a CP-ABE construction for arbitrary circuits of bounded size that achieves selective security in the CP-ABE security game [GPSW06, BSW07] under the MDDH assumption for suitable k polynomially related to the bound on circuit size.*

Proof. This follows by considering a variant of a Universal Circuit U_x where $U(C) = C(x)$, where C is a canonical representation of an arbitrary bounded-size circuit by a bounded-size string. In the CP-ABE setting, keys correspond to specific inputs x, and ciphertexts correspond to circuits C. We implement this by using our construction, and providing keys corresponding to the circuit U_x. Thus, when a key is used to decrypt a ciphertext corresponding to a circuit description C, the user will be able to decrypt iff $U_x(C) = C(x) = 1$, as desired. □

The remainder of this section contains a proof of Theorem 4.1.

Proof of Theorem 4.1. Our proof follows the "Move Forward and Shift" paradigm that was described in the Introduction. For intuition on how this works, please refer to the "Our Techniques" section of the Introduction. Below, we provide the mathematical details behind the proof.

Init. $\mathcal{B}$ first receives the $\ell+1$-multilinear problem where it is given the group description $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ and an problem instance $g, g^s, g^{c_1}, \ldots, g^{c_k}, T$. T is either $g_k^{s\prod_{j\in[1,k]} c_j}$ or a random group element in $\mathbb{G}_k$. (Note we slightly changed the variable names in the problem instance to better suit our proof.)

Next, the attacker declares the challenge input $x^* \in \{0,1\}^n$.

Setup. $\mathcal{B}$ chooses random $y_1, \ldots, y_n \in \mathbb{Z}_p$. For $i \in [1,n]$ set

$$h_i = \begin{cases} g^{y_i} & \text{if } x_i^* = 1 \\ g^{y_i + c_1} & \text{if } x_i^* = 0 \end{cases}$$

Remark. Note that over $\mathbb{Z}_p$, the above choices of h_i are distributed identically with the "real life" distribution. More generally, what we need is that g^{y_i} is statistically close to, or indistinguishable from, $g^{y_i+c_1}$.

Next, $\mathcal{B}$ sets $g_k^{\alpha} = g_k^{\xi+\prod_{i\in[1,k]} c_i}$, where ξ is chosen randomly. It computes this using $g^{c_1}, \ldots, g^{c_k}$ from the assumption, by means of the iterated use of the pairing function.

Remark. Here we need that $g_k^{\xi+\prod_{i\in[1,k]} c_i}$ is statistically close to, or indistinguishable from, g_k^{ξ}. This holds perfectly over $\mathbb{Z}_p$.

Challenge Ciphertext. Let $S^* \subseteq [1,n]$ be the set of input indices where $x_i^* = 1$. $\mathcal{B}$ creates the challenge ciphertext as:

$$\mathrm{CT} = (T \cdot g_k^{s\xi},\ g^s,\ \forall j \in S^*\ C_j = (g^s)^{y_j})$$

If $T = g_k^{s\prod_{j\in[1,k]} c_j}$ then this is an encryption of 1; otherwise if T was chosen random in $\mathbb{G}_k$ then w.h.p. it is an encryption of 0.

KeyGen Phase. Both key generation phases are executed in the same manner by the reduction algorithm. Therefore, we describe them once here. The attacker will give a circuit $f = (n, q, A, B, \texttt{GateType})$ to the reduction algorithm such that $f(x^*) = 0$.

We can think of the proof as having some invariant properties on the depth of the gate we are looking at. Consider a gate w at depth j and the simulators viewpoint (symbolically) of r_w. If $f_w(x^*) = 0$, then the simulator will view r_w as the term $c_1 \cdot c_2 \cdots c_{j+1}$ plus some additional known randomization terms. If $f_w(x^*) = 1$, then the simulator will view r_w as the 0 plus some additional known randomization terms. If we can keep this property intact for simulating the keys up the circuit, the simulator will view r_{n+q} as $c_1 \cdot c_2 \cdots c_k$. This will allow for it to simulate the header component K_H by cancellation.

We describe how to create the key components for each wire w. Again, we organize key component creation into input wires, OR gates, and AND gates.

- *Input wire*
  Suppose $w \in [1,n]$ and is therefore by convention an input wire.

  If $(x^*)_w = 1$ then we choose r_w and z_w at random (as is done honestly). The key components are:

  $$(K_{w,1} = g^{r_w} h_w^{z_w},\ K_{w,2} = g^{z_w})$$

  If $(x^*)_w = 0$ then we let $r_w = c_1 c_2 + \eta_i$ and $z_w = -c_2 + \nu_i$, where η_i and ν_i are randomly chosen elements. The key components are:

  $$(K_{w,1} = g^{c_1 c_2 + \eta_w} h_w^{-c_2+\nu_w},\ K_{w,2} = g^{-c_2+\nu_w}) = (g^{-c_2 y_w + \eta_w + (y_w + c_1)\nu_w}, g^{-c_2+\nu_w})$$

Note a cancellation occurred that allowed for the first term to be computed. Observe that in both of these values are simulated consistent with our invariant.

Remark. Here we need that $g^{-c_2 y_w + \eta_w + (y_w + c_1)\nu_w}$ is appropriately close to a randomly chosen element. This holds perfectly over $\mathbb{Z}_p$.

- *OR gate*
  Now we consider a wire $w \in$ Gates and that $\texttt{GateType}(w) =$ OR. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w. If $f_w(x^*) = 1$, then we simply set a_w, b_w, r_w at random to values chosen by $\mathcal{B}$. Then the algorithm creates key components:

$$K_{w,1} = g^{a_w},\ K_{w,2} = g^{b_w},\ K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)}},\ K_{w,4} = g_j^{r_w - b_w \cdot r_{B(w)}}$$

  If $f_w(x^*) = 0$, then we set $a_w = c_{j+1} + \psi_w$ and $b_w = c_{j+1} + \phi_w$ and $r_w = c_1 \cdot c_2 \cdots c_{j+1} + \eta_w$, where ψ_w, ϕ_w, η_w are chosen randomly. Then the algorithm creates key components:

$$K_{w,1} = g^{c_{j+1} + \psi_w},\ K_{w,2} = g^{c_{j+1} + \phi_w},$$

$$K_{w,3} = g_j^{\eta_w - c_{j+1}\eta_{A(w)} - \psi_w(c_1 \cdots c_j + \eta_{A(w)})},\ K_{w,4} = g_j^{\eta_w - c_{j+1}\eta_{B(w)} - \phi_w(c_1 \cdots c_j + \eta_{B(w)})}$$

  $\mathcal{B}$ is able to create the last two key components due to a cancellation. Since both the $A(w)$ and $B(w)$ gates evaluated to 0 we had $r_{A(w)} = c_1 \cdots c_j + \eta_{A(w)}$ and similarly for $r_{B(w)}$. Note that computing $g_j^{c_1 \cdots c_j}$ is possible using the multi-linear maps.

  Remark. Here we need that $g_j^{\eta_w - \psi_w(c_1 \cdots c_j)}$ is appropriately close to a randomly chosen element (the given terms dominate the others). This holds perfectly over $\mathbb{Z}_p$.

- *AND gate*
  Now we consider a wire $w \in$ Gates and that $\texttt{GateType}(w) =$ OR. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w.

  If $f_w(x^*) = 1$, then we simply set a_w, b_w, r_w at random to values known by $\mathcal{B}$. Then the algorithm creates key components:

$$K_{w,1} = g^{a_w},\ K_{w,2} = g^{b_w},\ K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

  If $f_w(x^*) = 0$ and $f_{A(w)}(x^*) = 0$, then $\mathcal{B}$ sets $a_w = c_{j+1} + \psi_w, b_w = \phi_w$ and $r_w = c_1 \cdot c_2 \cdots c_{j+1} + \eta_w$, where ψ_w, ϕ_w, η_w are chosen randomly. Then the algorithm creates key components:

$$K_{w,1} = g^{c_{j+1} + \psi_w},\ K_{w,2} = g^{\phi_w},\ K_{w,3} = g_j^{\eta_w - \psi_w c_1 \cdots c_j - (c_{j+1} + \psi_w)\eta_{A(w)} - \phi_w(r_{B(w)})}$$

  $\mathcal{B}$ can create the last component due to cancellation. Since the $A(w)$ gate evaluated to 0, we have $r_{A(w)} = c_1 \cdot c_2 \cdots c_j + \eta_{A(w)}$. Note that $g_j^{r_{B(w)}}$ is always computable regardless of whether $f_{A(w)}(x^*)$ evaluated to 0 or 1, since $g_j^{c_1 \cdots c_j}$ is always computable using the multilinear maps.
  The case where $f_{B(w)}(x^*) = 0$ and $f_{A(w)}(x^*) = 1$ is performed in a symmetric to what is above, with the roles of a_w and b_w reversed.

  Remark. Here we need that $g_j^{\eta_w - (\psi_w + \phi_w) \cdot (c_1 \cdots c_j)}$ is appropriately close to a randomly chosen element (the given terms dominate the others). This holds perfectly over $\mathbb{Z}_p$.

For the output gate we chose η_w at random. Thus, at the end we have $r_{n+q} = \prod_{i \in [1,k]} c_i + \eta_{n+q}$ for the output gate. This gives us a final cancellation in computing the "header" component of the key as $K_H = (g_{k-1})^{\alpha - r_{n+q}} = (g_{k-1})^{\xi - \eta_w}$.

Guess. $\mathcal{B}$ receives back the guess $M' \in \{0,1\}$ of the message from $\mathcal{A}$. If $M' = 1$ it guesses that T is a tuple; otherwise, it guesses that it is random.

This immediately shows that any adversary with non-trivial advantage in the KP-ABE selective security game will have an identical advantage in breaking the k-MDDH assumption. □

References

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

[ABV+12] Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Functional encryption for threshold functions (or fuzzy ibe) from lattices. In *Public Key Cryptography*, pages 280–297, 2012.

[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.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. Cryptology ePrint Archive, Report 2012/265, 2012. `http://eprint.iacr.org/`.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, 2006.

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

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[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.

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

[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.

[GGH12] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices and applications. *IACR Cryptology ePrint Archive*, 2012:610, 2012.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GKP+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. In *STOC*, 2013.

[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.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits. In *STOC*, 2013.

[Ham11] Mike Hamburg. Spatial encryption. *IACR Cryptology ePrint Archive*, 2011:389, 2011.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

[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.

[LW11] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588, 2011.

[LW12] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, pages 180–198, 2012.

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

[OT12] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In *EUROCRYPT*, pages 591–608, 2012.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[Rot12] Ron Rothblum. On the circular security of bit-encryption. Cryptology ePrint Archive, Report 2012/102, 2012. `http://eprint.iacr.org/`.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM Conference on Computer and Communications Security*, pages 463–472, 2010.

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

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

[SW12] Amit Sahai and Brent Waters. Attribute-based encryption for circuits from multilinear maps. *IACR Cryptology ePrint Archive*, 2012:592, 2012.

[Wat12] Brent Waters. Functional encryption for regular languages. In *CRYPTO*, pages 218–235, 2012.

A Definitions for ABE for Circuits

We now give a formal definition of our Attribute-Based Encryption for circuits. Our definition is fit for bounded circuits.

Setup$(1^\lambda, n, \ell)$ The setup algorithm takes as input the security parameter, the length n of input descriptors from the ciphertext and a bound ℓ on the circuit depth. It outputs the public parameters PP and a master key MSK.

Encrypt$(\text{PP}, x \in \{0,1\}^n, M)$ The encryption algorithm takes as input the public parameters PP, a bit string $x \in \{0,1\}^n$ representing the assignment of boolean variables, and a message m. It outputs a ciphertext CT.

Key Generation$(\text{MSK}, f = (n, q, A, B, \texttt{GateType}))$ The key generation algorithm takes as input the master key MSK and a description of a circuit f according to the conventions established in Section 2, where the depth of f is at most ℓ. 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 $x \in \{0,1\}^n$, and depth ℓ circuits f where $f(x) = 1$. If $\mathit{Encrypt}(\text{PP}, x, M) \to \text{CT}$ and $\mathit{KeyGen}(\text{MSK}, f) \to \text{SK}$ where PP, MSK were generated from a call to the setup algorithm, then $\mathit{Decrypt}(\text{SK}, \text{CT}) = M$.

Security Model for ABE for circuits. We now describe a game-based security definition for ABE for circuits. As in other similar systems (e.g. [BF03, SW05, GPSW06]), 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 circuit f of his choice, but must encrypt to some string x^* such that every circuit f for which a private key was requested for we have $f(x^*) = 0$. 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 circuit descriptions f of its choice. The challenger returns $\mathit{KeyGen}(\text{MSK}, f)$.

Challenge. The adversary submits two equal length messages M_0 and M_1. In addition, the adversary gives a challenge string x^* such that for all f requested in Phase 1 we have that $f(x^*) = 0$. Then the challenger flips a random coin $b \in \{0,1\}$, and computes $\mathit{Encrypt}(\text{PP}, x^*, M) \to \text{CT}^*$. The challenge ciphertext CT^* is given to the adversary.

Phase 2. Phase 1 is repeated with the restriction that for all f requested $f(x^*) = 0$.

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 A.1. *An attribute-based encryption scheme for circuits is secure if all polynomial time adversaries have at most a negligible advantage in the above game.*

Definition A.2. *We say that a system is* selectively *secure if the system is secure in a game where we add an Init stage before setup where the adversary commits to the challenge string x^*.*

B Our Construction: based on GGH graded algebras

We now describe how to modify our construction to use the GGH [GGH12] graded algebras analogue of multilinear maps. The translation of our scheme above is straightforward to the GGH setting. We start by providing background on Garg et al.'s lattice-based "approximate" multilinear maps (a.k.a. "graded encoding systems") [GGH12].

B.1 Graded Encoding Systems: Definition

Garg, Gentry and Halevi (GGH) [GGH12] defined an "approximate" version of a multilinear group family, which they call a *graded encoding system*. As a starting point, they view g_i^α in a multilinear group family as simply an *encoding* of α at "level-i". This encoding permits basic functionalities, such as equality testing (it is easy to check that two level-i encodings encode the same exponent), additive homomorphism (via the group operation in $\mathbb{G}_i$), and bounded multiplicative homomorphism (via the multilinear map e). They retain the notion of a somewhat homomorphic encoding with equality testing, but they use probabilistic encodings, and replace the multilinear group family with "less structured" sets of encodings related to lattices.

Abstractly, their n-graded encoding system for a ring R includes a system of sets $\mathcal{S} = \{S_i^{(\alpha)} \subset \{0,1\}^* : i \in [0,n], \alpha \in R\}$ such that, for every fixed $i \in [0,n]$, the sets $\{S_i^{(\alpha)} : \alpha \in R\}$ are disjoint (and thus form a partition of $S_i \stackrel{\text{def}}{=} \bigcup_\alpha S_i^{(\alpha)}$). The set $S_i^{(\alpha)}$ consists of the "level-i encodings of α". Moreover, the system comes equipped with efficient procedures, as follows:[6]

Instance Generation. The randomized $\mathsf{InstGen}(1^\lambda, 1^n)$ takes as input the security parameter λ and integer n. The procedure outputs $(\mathsf{params}, \mathbf{p}_{zt})$, where params is a description of an n-graded encoding system as above, and $\mathbf{p}_{zt}$ is a level-n "zero-test parameter".

Ring Sampler. The randomized $\mathsf{samp}(\mathsf{params})$ outputs a "level-zero encoding" $a \in S_0$, such that the induced distribution on α such that $a \in S_0^{(\alpha)}$ is statistically uniform.

Encoding. The (possibly randomized) $\mathsf{enc}(\mathsf{params}, i, a)$ takes $i \in [n]$ and a level-zero encoding $a \in S_0^{(\alpha)}$ for some $\alpha \in R$, and outputs a level-i encoding $u \in S_i^{(\alpha)}$ for the same α.

[6] Since GGH's realization of a graded encoding system uses "noisy" encodings over ideal lattices, the procedures incorporate information about the magnitude of the noise.

Re-Randomization. The randomized $\mathsf{reRand}(\mathsf{params}, i, u)$ re-randomizes encodings to the same level, as long as the initial encoding is under a given noise bound. Specifically, for a level $i \in [n]$ and encoding $u \in S_i^{(\alpha)}$, it outputs another encoding $u' \in S_i^{(\alpha)}$. Moreover for any two encodings $u_1, u_2 \in S_i^{(\alpha)}$ whose noise bound is at most some b, the output distributions of $\mathsf{reRand}(\mathsf{params}, i, u_1)$ and $\mathsf{reRand}(\mathsf{params}, i, u_2)$ are statistically the same.

Addition and negation. Given params and two encodings at the same level, $u_1 \in S_i^{(\alpha_1)}$ and $u_2 \in S_i^{(\alpha_2)}$, we have $\mathsf{add}(\mathsf{params}, u_1, u_2) \in S_i^{(\alpha_1+\alpha_2)}$, and $\mathsf{neg}(\mathsf{params}, u_1) \in S_i^{(-\alpha_1)}$, subject to bounds on the noise.

Multiplication. For $u_1 \in S_{i_1}^{(\alpha_1)}$, $u_2 \in S_{i_2}^{(\alpha_2)}$, we have $\mathsf{mult}(\mathsf{params}, u_1, u_2) \in S_{i_1+i_2}^{(\alpha_1 \cdot \alpha_2)}$.

Zero-test. The procedure $\mathsf{isZero}(\mathsf{params}, \mathbf{p}_{zt}, u)$ outputs 1 if $u \in S_n^{(0)}$ and 0 otherwise. Note that in conjunction with the procedure for subtracting encodings, this gives us an equality test.

Extraction. This procedure extracts a "canonical" and "random" representation of ring elements from their level-n encoding. Namely $\mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u)$ outputs (say) $K \in \{0,1\}^\lambda$, such that:

(a) With overwhelming probability over the choice of $\alpha \in R$, for any two $u_1, u_2 \in S_n^{(\alpha)}$, $\mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u_1) = \mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u_2)$,

(b) The distribution $\{\mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u) : \alpha \in R, u \in S_n^{(\alpha)}\}$ is statistically uniform over $\{0,1\}^\lambda$.

We can extend add and mult to handle more than two encodings as inputs, by applying the binary versions of add and mult iteratively. Also, we use the canonicalizing encoding algorithm (as defined in Remark 2 of [GGH12]) $\mathsf{cenc}_\ell(\mathsf{params}, i, a)$ which takes as input encoding of a and generates another encoding according to a "nice" distribution. This parameter ℓ essentially captures the noise present in the encodings. In our scheme the maximum value ℓ takes will be a small constant.

Recall that the k-multilinear assumption for the graded encodings as follows:

Assumption B.1 (k-Graded Multilinear Decisional Diffie-Hellman Assumption: k-GMDDH [GGH12])**.** *The k-GMDDH problem states the following: A challenger runs $(\mathsf{params}, \mathbf{p}_{zt}) \leftarrow \mathsf{InstGen}(1^\lambda, 1^k)$. Then it picks at random $s, c_1, \ldots, c_k \leftarrow \mathsf{samp}(\mathsf{params})$.*

The assumption then states that given $\mathsf{cenc}_1(\mathsf{params}, 1, s), \mathsf{cenc}_1(\mathsf{params}, 1, c_1), \ldots, \mathsf{cenc}_1(\mathsf{params}, 1, c_k)$, it is hard to distinguish $T = \mathsf{cenc}_1(\mathsf{params}, k, s\prod_{j\in[1,k]} c_j)$ from $T = \mathsf{cenc}_1(\mathsf{params}, k, \mathsf{samp}(\mathsf{params}))$, with better than negligible advantage (in security parameter λ).

B.2 Graded Encoding Systems: Realization

Concretely, GGH's n-graded encoding system works as follows. (This is a whirlwind overview; see [GGH12] for details.) The system uses three rings. First, it uses the ring of integers $\mathcal{O}$ of the m-th cyclotomic field. This ring is typically represented as the ring of polynomials $\mathcal{O} = \mathbb{Z}[x]/(\Phi_m(x))$, where $\Phi_m(x)$ is the m-th cyclotomic polynomial, which has degree $N = \phi(m)$. Second, for some suitable integer modulus q, it uses the quotient ring $\mathcal{O}/(q) = \mathbb{Z}_q[x]/(\Phi_m(x))$, similar to the NTRU encryption scheme [HPS98]. The encodings live in $\mathcal{O}/(q)$. Finally, it uses the quotient ring $R = \mathcal{O}/\mathcal{I}$, where $\mathcal{I} = \langle g \rangle$ is a principal ideal of $\mathcal{O}$ that is generated by g and where $|\mathcal{O}/\mathcal{I}|$ is a large prime. This is the ring "R" referred to above; elements of R are what is encoded.

What does a GGH encoding look like? For a fixed random $z \in \mathcal{O}/(q)$, an element of $S_i^{(\alpha)}$ – that is, a level-i encoding of $\alpha \in R$ – has the form $e/z^i \in \mathcal{O}/(q)$, where $e \in \mathcal{O}$ is a "small" representative

of the coset $\alpha + \mathcal{I}$ (it has coefficients that are very small compared to q). To add encodings $e_1/z^i \in S_i^{(\alpha_1)}$ and $e_2/z^i \in S_i^{(\alpha_2)}$, just add them in $\mathcal{O}/(q)$ to obtain $(e_1+e_2)/z^i$, which is in $S_i^{(\alpha_1+\alpha_2)}$ if $e_1 + e_2$ is "small". To mult encodings $e_1/z^{i_1} \in S_{i_1}^{(\alpha_1)}$ and $e_2/z^{i_2} \in S_{i_2}^{(\alpha_2)}$, just multiply them in $\mathcal{O}/(q)$ to obtain $e_1 \cdot e_2/z^{i_1+i_2}$, which is in $S_{i_1+i_2}^{(\alpha_1 \cdot \alpha_2)}$ if $e_1 \cdot e_2$ is "small". This smallness condition limits the GGH encoding system to degree polynomial in the security parameter. Intuitively, dividing encodings does not "work", since the resulting denominator has a nontrivial term that is not z.

The GGH params allow everyone to generate encodings of random (known) values. The params include a level-1 encoding of 1 (from which one can generate encodings of 1 at other levels), and (for each $i \in [n]$) a sufficient number of level-i encodings of 0 to enable re-randomization. To encode (say at level-1), run $\mathsf{samp}(\mathsf{params})$ to sample a small element a from $\mathcal{O}$, e.g. according to a discrete Gaussian distribution. For a Gaussian with appropriate deviation, this will induce a statistically uniform distribution over the cosets of $\mathcal{I}$. Then, multiply a with the level-1 encoding of 1 to get a level-1 encoding u of $a \in R$. Finally, run $\mathsf{reRand}(\mathsf{params}, 1, u)$, which involves adding a random Gaussian linear combination of the level-1 encodings of 0, whose noisiness (i.e., numerator size) "drowns out" the initial encoding. The parameters for the GGH scheme can be instantiated such that the re-randomization procedure can be used for any pre-specified polynomial number of times.

To permit testing of whether a level-n encoding $u = e/z^n \in S_n$ encodes 0, GGH publishes a level-n zero-test parameter $\mathbf{p}_{zt} = hz^n/g$, where h is "somewhat small"[7] and g is the generator of $\mathcal{I}$. The procedure $\mathsf{isZero}(\mathsf{params}, \mathbf{p}_{zt}, u)$ simply computes $\mathbf{p}_{zt} \cdot u$ and tests whether its coefficients are small modulo q. If u encodes 0, then $e \in \mathcal{I}$ and equals $g \cdot c$ for some (small) c, and thus $\mathbf{p}_{zt} \cdot u = h \cdot c$ has no denominator and is small modulo q. If u encodes something nonzero, $\mathbf{p}_{zt} \cdot u$ has g in the denominator and is not small modulo q. The $\mathsf{ext}(\mathsf{params}, \mathbf{p}_{zt}, u)$ procedure works by applying a strong extractor to the most significant bits of $\mathbf{p}_{zt} \cdot u$. For any two $u_1, u_2 \in S_n^{(\alpha)}$, we have (subject to noise issues) $u_1 - u_2 \in S_n^{(0)}$, which implies $\mathbf{p}_{zt}(u_1 - u_2)$ is small, and hence $\mathbf{p}_{zt} \cdot u_1$ and $\mathbf{p}_{zt} \cdot u_2$ have the same most significant bits (for an overwhelming fraction of α's).

Garg et al. provide an extensive cryptanalysis of the encoding system, which we will not review here. We remark that the underlying assumptions are stronger, but related to, the hardness assumption underlying the NTRU encryption scheme: that it is hard to distinguish a uniformly random element from $\mathcal{O}/(q)$ from a ratio of "small" elements – i.e., an element $u/v \in \mathcal{O}/(q)$ where $u, v \in \mathcal{O}/(q)$ both have coefficients that are on the order of (say) q^ϵ for small constant ϵ.

B.3 Our Construction

Now we provide our construction in GGH's n-graded encoding system. **For ease of notation on the reader, we suppress repeated params arguments that are provided to every algorithm.**. Thus, for instance, we will write $\alpha \leftarrow \mathsf{samp}()$ instead of $\alpha \leftarrow \mathsf{samp}(\mathsf{params})$. Note that in our scheme, there will only ever be a single uniquely chosen value for params throughout the scheme, so there is no cause for confusion.

$\mathbf{Setup}(1^\lambda, n, \ell)$. The setup algorithm takes as input, a security parameter λ, the maximum depth ℓ of a circuit, and the number of boolean inputs n.

It then runs $(\mathbf{p}_{zt}) \leftarrow \mathsf{InstGen}(1^\lambda, 1^{k=\ell+1})$. Recall that params will be implicitly given as input to all GGH-related algorithms below. Next, it samples $\alpha, \hat{h_1}, \ldots, \hat{h_n} \leftarrow \mathsf{samp}()$.

[7] Its coefficients are on the order of (say) $q^{2/3}$, while other terms – such as a numerator e or the principal ideal generator g – are much, much smaller.

The public parameters, PP, consist of $\mathbf{p}_{zt}$, plus:

$$H = \mathsf{cenc}_2(k, \alpha), h_1 = \mathsf{cenc}_2(1, \hat{h_1}), \ldots, h_n = \mathsf{cenc}_2(1, \hat{h_n})$$

The master secret key MSK is α.

Encrypt($\mathrm{PP}, x \in \{0,1\}^n, M \in \{0,1\}$**).** The encryption algorithm takes in the public parameters, an descriptor input $x \in \{0,1\}^n$, and a message bit $M \in \{0,1\}$.

The encryption algorithm chooses a random $s \leftarrow \mathsf{samp}()$. If $M = 0$ it sets C_M to be a random value:

$$C_M = \mathsf{cenc}_3(k, \mathsf{samp}())$$

otherwise it lets

$$C_M = \mathsf{cenc}_3(k, H \cdot s)$$

Next, let S be the set such of i such that $x_i = 1$.

The ciphertext is created as

$$\mathrm{CT} = (C_M,\ \tilde{s} = \mathsf{cenc}_1(1, s),\ \forall i \in S\ \ C_i = \mathsf{cenc}_3(1, h_i \cdot s))$$

KeyGen($\mathrm{MSK} = \alpha, f = (n, q, A, B, \mathtt{GateType})$**).** The algorithm takes in the master secret key and a description f of a circuit. Recall, that the circuit has $n + q$ wires with n input wires, q gates and the wire $n + q$ designated as the output wire.

The key generation algorithm chooses random $r_1, \ldots, r_{n+q} \leftarrow \mathsf{samp}()$, where we think of randomness r_w as being associated with wire w. The algorithm produces a "header" component

$$K_H = \mathsf{cenc}_3(k - 1, \alpha - r_{n+q})$$

Next, the algorithm generates key components for every wire w. The structure of the key components depends upon if w is an input wire, an OR gate, or an AND gate. We describe how it generates components for each case.

- *Input wire*
  By our convention if $w \in [1, n]$ then it corresponds to the w-th input. The key generation algorithm chooses random $z_w \leftarrow \mathsf{samp}()$.

  The key components are:

  $$K_{w,1} = \mathsf{cenc}_3(1, \mathsf{enc}(1, r_w) + h_w \cdot z_w),\ K_{w,2} = \mathsf{cenc}_3(1, -z_w)$$

- *OR gate*
  Suppose that wire $w \in$ Gates and that $\mathtt{GateType}(w) = \mathrm{OR}$. In addition, let $j = \mathtt{depth}(w)$ be the depth of wire w. The algorithm will choose random $a_w, b_w \leftarrow \mathsf{samp}()$. Then the algorithm creates key components:

  $$K_{w,1} = \mathsf{cenc}_3(1, a_w),\ K_{w,2} = \mathsf{cenc}_3(1, b_w),$$

  $$K_{w,3} = \mathsf{cenc}_3(j, r_w - a_w \cdot r_{A(w)}),\ K_{w,4} = \mathsf{cenc}_3(j, r_w - b_w \cdot r_{B(w)})$$

- *AND gate*
  Suppose that wire $w \in$ Gates and that $\texttt{GateType}(w) = \text{AND}$. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w. The algorithm will choose random $a_w, b_w \leftarrow \mathsf{samp}()$.

$$K_{w,1} = \mathsf{cenc}_3(1, a_w),\ K_{w,2} = \mathsf{cenc}_3(1, b_w),$$

$$K_{w,3} = \mathsf{cenc}_3(j, r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)})$$

We will sometimes refer to the $K_{w,3}, K_{w,4}$ of the AND and OR gates as the "shift" components. This terminology will take on more meaning when we see how they are used during decryption.

The secret key SK output consists of the description of f, the header component K_H and the key components for each wire w.

Decrypt(SK, CT)**.** Suppose that we are evaluating decryption for a secret key associated with a circuit $f = (n, q, A, B, \texttt{GateType})$ and a cipherext with input x. We will be able to decrypt if $f(x) = 1$.

We begin by observing that the goal of decryption should be to compute a level k encoding of $\alpha \cdot s$ such that we can test if this is equal to C_M. First, there is a header computation where we compute $E' = K_H \cdot \tilde{s}$ Note that E' should thus be a level k encoding of $\alpha s - r_{n+q} \cdot s$. Our goal is now reduced to computing a level k encoding of $r_{n+q} \cdot s$.

Next, we will evaluate the circuit from the bottom up. Consider wire w at depth j; if $f_w(x) = 1$ then, our algorithm will compute E_w to be a level $j+1$ encoding of sr_w. Note that if $f_w(x) = 0$ nothing needs to be computed for that wire, since we have a monotonic circuit. Our decryption algorithm proceeds iteratively starting with computing E_1 and proceeds in order to finally compute E_{n+q}. Computing these values in order ensures that the computation on a depth $j-1$ wire (that evaluates to 1) will be defined before computing for a depth j wire. We show how to compute E_w for all w where $f_w(x) = 1$, again breaking the cases according to whether the wire is an input, AND or OR gate.

- *Input wire*
  By our convention if $w \in [1, n]$ then it corresponds to the w-th input. Suppose that $x_w = f_w(x) = 1$. The algorithm computes:

$$E_w = K_{w,1} \cdot \tilde{s} + K_{w,2} \cdot C_w$$

  Thus, E_w computes a level 2 encoding of $(r_w + \hat{h_w} \cdot z_w) \cdot s + (-z_w) \cdot \hat{h_w} \cdot s = sr_w$.

- *OR gate*
  Consider a wire $w \in$ Gates and that $\texttt{GateType}(w) = \text{OR}$. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w. Suppose that $f_w(x) = 1$. If $f_{A(w)}(x) = 1$ (the first input evaluated to 1) then we compute:

$$E_w = E_{A(w)} \cdot K_{w,1} + K_{w,3} \cdot \tilde{s}$$

  Thus, E_w computes a level $j+1$ encoding of $sr_{A(w)} \cdot a_w + (r_w - a_w \cdot r_{A(w)}) \cdot s = sr_w$.

  Alternatively, if $f_{A(w)}(x) = 0$, but $f_{B(w)}(x) = 1$, then we compute:

$$E_w = E_{B(w)} \cdot K_{w,2} + K_{w,4} \cdot \tilde{s}$$

  This similarly computes a level $j+1$ encoding of $sr_{B(w)} \cdot b_w + (r_w - b_w \cdot r_{B(w)}) \cdot s = sr_w$.

Let's examine this mechanism for the case where the first input is 1 ($f_{A(w)}(x) = 1$). In this case the algorithm "moves" the value $E_{A(w)}$ from level j to level $j+1$ when multiplying it with $K_{w,1}$. It then adds it to $K_{w,3} \cdot \tilde{s}$ which "shifts" that result to E_w.

Suppose that $f_{A(w)}(x) = 1$, but $f_{B(w)}(x) = 0$. A critical feature of the mechanism is that an attacker cannot perform a "backtracking" attack to compute $E_{B(w)}$. The reason is that the GGH encoding cannot be reversed to go from level $j+1$ to level j. (See [GGH12] for details on why this is the case.) If this were not the case, it would be debilitating for security as gate $B(w)$ might have fanout greater than 1. This type of backtracking attacking is why existing ABE constructions are limited to circuits with fanout of 1.

- *AND gate*
  Consider a wire $w \in$ Gates and that $\texttt{GateType}(w) = \text{AND}$. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w. Suppose that $f_w(x) = 1$. Then $f_{A(w)}(x) = f_{B(w)}(x) = 1$ and we compute:
  $$E_w = E_{A(w)} \cdot K_{w,1} + E_{B(w)} \cdot K_{w,2} + K_{w,3} \cdot \tilde{s}$$
  Note that this computes a level $j+1$ encoding of sr_w in a manner analogous to above.

If $f(x) = f_{n+q}(x) = 1$, then the algorithm will compute E_{n+q} to be a level k encoding of $r_{n+q} \cdot s$. It finally computes $E' + E_{n+q}$ which is a level k encoding of αs and tests if this equals C_M using $\mathsf{isZero}(\mathbf{p}_{zt}, E' + E_{n+q} - C_M)$, outputting $M = 1$ if so and $M = 0$ otherwise. Correctness holds with high probability.

A Quick Remark about Message Length. Our encryption algorithm takes as input a single bit message. We can extend this to longer messages using the ext algorithm provided by the GGH encoding [GGH12]. We keep to single bit messages for clarity of the scheme and proof of security.

C Proof of Security in GGH framework

We prove (selective) security in the security model given by GPSW [GPSW06], where the key access structures are monotonic circuits. For a circuit of max depth $k-1$ we prove security under the GGH analog of the decision k-multilinear assumption.

We show that if there exist a poly-time attacker $\mathcal{A}$ on our ABE system for circuits of depth ℓ and inputs of length n in the selective security game then we can construct a poly-time algorithm on the GGH-analog of the decision $\ell+1$-multilinear assumption with non-negligible advantage. We describe how $\mathcal{B}$ interacts with $\mathcal{A}$.

Theorem C.1. *The construction given in the previous section achieves selective security for arbitrary circuits of depth $k-1$ in the KP-ABE security game under the k-GMDDH assumption.*

By using universal circuits, we obtain as an immediate corollary:

Corollary C.2. *There exists a CP-ABE construction for arbitrary circuits of bounded size that achieves selective security in the CP-ABE security game [GPSW06, BSW07] under the GMDDH assumption for suitable k polynomially related to the bound on circuit size.*

Proof. This follows by considering a variant of a Universal Circuit U_x where $U(C) = C(x)$, where C is a canonical representation of an arbitrary bounded-size circuit by a bounded-size string. In the CP-ABE setting, keys correspond to specific inputs x, and ciphertexts correspond to circuits

C. We implement this by using our construction, and providing keys corresponding to the circuit U_x. Thus, when a key is used to decrypt a ciphertext corresponding to a circuit description C, the user will be able to decrypt iff $U_x(C) = C(x) = 1$, as desired. □

The remainder of this section contains a proof of Theorem 4.1.

Proof of Theorem C.1. Our proof follows the "Move Forward and Shift" paradigm that was described in the Introduction. For intuition on how this works, please refer to the "Our Techniques" section of the Introduction. Below, we provide the mathematical details behind the proof.

Init. $\mathcal{B}$ first receives the $k = \ell + 1$-multilinear problem where it is given the encoding description $\mathbf{p}_{zt}$ and a problem instance consisting of random level 1 encodings, $\tilde{s} = \mathsf{cenc}_1(1, s), \tilde{c_1} = \mathsf{cenc}_1(1, c_1), \ldots, \tilde{c_k} = \mathsf{cenc}_1(1, c_k)$, where $s, c_1, \ldots, c_k \leftarrow \mathsf{samp}()$, and a level k encoding T. This last encoding T is either $\mathsf{cenc}_1(k, s\prod_{j\in[1,k]} c_j)$ or a random encoding $\mathsf{cenc}_1(k, \mathsf{samp}())$.

Next, the attacker declares the challenge input $x^* \in \{0,1\}^n$.

Setup. $\mathcal{B}$ chooses random $y_1, \ldots, y_n \in \mathbb{Z}_p$. For $i \in [1, n]$ set

$$h_i = \begin{cases} \mathsf{cenc}_2(1, \mathsf{enc}(1, y_i)) & \text{if } x^*_i = 1 \\ \mathsf{cenc}_2(1, \mathsf{enc}(1, y_i) + \tilde{c_1}) & \text{if } x^*_i = 0 \end{cases}$$

Remark. Note that due to rerandomization, the above choices of h_i are (jointly) distributed within negligible statistical distance to the "real life" distribution.

Next, $\mathcal{B}$ sets H, which should be a (rerandomized) level k encoding of α, to be $\mathsf{cenc}_2(k, \mathsf{enc}(k, \xi) + \prod_{i\in[1,k]} \tilde{c_i})$, where $\xi \leftarrow \mathsf{samp}()$ is chosen randomly.

Remark. Again, due to rerandomization and the random choice of ξ, the value H above is distributed within negligible statistical distance to the "real life" distribution, conditioned on all other choices so far.

Challenge Ciphertext. Let $S^* \subseteq [1, n]$ be the set of input indices where $x^*_i = 1$. $\mathcal{B}$ creates the challenge ciphertext as:

$$\mathrm{CT} = (\mathsf{cenc}_3(k, T + \mathsf{enc}(k-1, \xi) \cdot \tilde{s}),\ \tilde{s},\ \forall j \in S^*\ C_j = \mathsf{cenc}_3(1, y_j \cdot \tilde{s}))$$

If T is an encoding of $s\prod_{j\in[1,k]} c_j$, then this challenge ciphertext is distributed within negligible statistical distance of an honestly generated encryption of 1; otherwise if T was chosen as a random level k encoding, then this ciphertext is distributed within negligible statistical distance of an honestly generated encryption of 0. This follows immediately from rerandomization and the choice of variables above.

KeyGen Phase. Both key generation phases are executed in the same manner by the reduction algorithm. Therefore, we describe them once here. The attacker will give a circuit $f = (n, q, A, B, \texttt{GateType})$ to the reduction algorithm such that $f(x^*) = 0$.

We can think of the proof as having some invariant properties on the depth of the gate we are looking at. Consider a gate w at depth j and the simulators viewpoint (symbolically) of r_w. If $f_w(x^*) = 0$, then the simulator will view r_w as the term $c_1 \cdot c_2 \cdots c_{j+1}$ plus some additional known randomization terms. If $f_w(x^*) = 1$, then the simulator will view r_w as zero plus some additional known randomization terms. If we can keep this property intact for simulating the keys up the

circuit, the simulator will view r_{n+q} as $c_1 \cdot c_2 \cdots c_k$. This will allow for it to simulate the header component K_H by cancellation.

We describe how to create the key components for each wire w. Again, we organize key component creation into input wires, OR gates, and AND gates.

- *Input wire*
  Suppose $w \in [1, n]$ and is therefore by convention an input wire.

  If $(x^*)_w = 1$ then we choose r_w and z_w at random using $\mathsf{samp}()$ (as is done honestly). The key components are:

$$K_{w,1} = \mathsf{cenc}_3(1, \mathsf{enc}(1, r_w) + h_w \cdot z_w),\ K_{w,2} = \mathsf{cenc}_3(1, \mathsf{enc}(1, -z_w))$$

  If $(x^*)_w = 0$ then we implicitly let $r_w = c_1 c_2 + \eta_i$ and $z_w = -c_2 + \nu_i$, where η_i and ν_i are randomly chosen elements using $\mathsf{samp}()$. The key components are computed as follows

$$K_{w,1} = \mathsf{cenc}_3(1, -\tilde{c_2} \cdot y_w + \mathsf{enc}(1, \eta_w) + \mathsf{enc}(1, y_w \cdot \nu_w) + \tilde{c_1} \cdot \nu_w)$$

$$K_{w,2} = \mathsf{cenc}_3(1, -\tilde{c_2} + \mathsf{enc}(1, \nu_w))$$

  Please refer to the previous proof for intuition about the calculations above.

  Remark. Note that these keys are distributed within negligible statistical distance to honestly generated keys due to rerandomization and the random choices of η_w and ν_w.

- *OR gate*
  Now we consider a wire $w \in$ Gates and that $\mathtt{GateType}(w) = \text{OR}$. In addition, let $j = \mathtt{depth}(w)$ be the depth of wire w. If $f_w(x^*) = 1$, then we simply set a_w, b_w, r_w at random to values chosen by $\mathcal{B}$ using $\mathsf{samp}()$. Then the algorithm creates key components honestly. Note that if $f_w(x^*) = 1$ then it must be that either $f_{A(w)}(x^*) = 1$ or $f_{B(w)}(x^*) = 1$. Below, we first consider the case that both $f_{A(w)}(x^*) = 1$ and $f_{B(w)}(x^*) = 1$

$$K_{w,1} = \mathsf{cenc}_3(1, \mathsf{enc}(1, a_w)),\ K_{w,2} = \mathsf{cenc}_3(1, \mathsf{enc}(1, b_w)),$$
$$K_{w,3} = \mathsf{cenc}_3(j, \mathsf{enc}(j, r_w - a_w \cdot r_{A(w)})),\ K_{w,4} = \mathsf{cenc}_3(j, \mathsf{enc}(j, r_w - b_w \cdot r_{B(w)}))$$

  Note that if $f_{A(w)}(x^*) = 0$ or $f_{B(w)}(x^*) = 0$, then the computation would be slightly different. If $f_{A(w)}(x^*) = 0$, then $r_{A(w)} = c_1 \cdots c_j + \eta_{A(w)}$. Thus, the computation of $K_{w,3}$ above would be:

$$K_{w,3} = \mathsf{cenc}_3(j, \mathsf{enc}(j, r_w) - a_w \cdot (\tilde{c_1} \cdots \tilde{(c_j)} + \mathsf{enc}(j, \eta_{A(w)})))$$

  If $f_{B(w)}(x^*) = 0$, then a similar calculation would be done for $K_{w,4}$.

  If $f_w(x^*) = 0$, then we implicitly set $a_w = c_{j+1} + \psi_w$ and $b_w = c_{j+1} + \phi_w$ and $r_w = c_1 \cdot c_2 \cdots c_{j+1} + \eta_w$, where ψ_w, ϕ_w, η_w are chosen randomly using $\mathsf{samp}()$. Then the algorithm creates key components as follows. Below, recall that $\mathbf{y}$ is a level 1 encoding of 1 that is included as part of the params that were provided as part of the assumption.

$$K_{w,1} = \mathsf{cenc}_3(1, \widetilde{c_{j+1}} + \mathsf{enc}(1, \psi_w)),\ K_{w,2} = \mathsf{cenc}_3(1, \widetilde{c_{j+1}} + \mathsf{enc}(1, \phi_w)),$$

$$K_{w,3} = \mathsf{cenc}_3(j, \mathsf{enc}(j, \eta_w) - \mathbf{y}^{j-1} \cdot \widetilde{c_{j+1}} \cdot \eta_{A(w)} - \psi_w \cdot (\tilde{c_1} \cdots \tilde{c_j} + \mathsf{enc}(j, \eta_{A(w)})))$$

$$K_{w,4} = \mathsf{cenc}_3(j, \mathsf{enc}(j, \eta_w) - \mathbf{y}^{j-1} \cdot \widetilde{c_{j+1}} \cdot \eta_{B(w)} - \phi_w \cdot (\tilde{c_1} \cdots \tilde{c_j} + \mathsf{enc}(j, \eta_{B(w)})))$$

For intuition regarding the calculation above, please see the previous proof.

Remark. Again, note that these keys are distributed within negligible statistical distance to honestly generated keys due to rerandomization and the random choices of η_w, ψ_w, and ϕ_w.

- *AND gate*
  Now we consider a wire $w \in$ Gates and that $\texttt{GateType}(w) = \text{OR}$. In addition, let $j = \texttt{depth}(w)$ be the depth of wire w.

  If $f_w(x^*) = 1$, then we simply set a_w, b_w, r_w at random to values known by $\mathcal{B}$ using $\mathsf{samp}()$. Then the algorithm creates key components honestly. Note that because both $f_{A(w)}(x^*) = 1$ and $f_{B(w)}(x^*) = 1$, below the $r_{A(w)}$ and $r_{B(w)}$ are fully known by $\mathcal{B}$.

$$K_{w,1} = \mathsf{cenc}_3(1, \mathsf{enc}(1, a_w)),\ K_{w,2} = \mathsf{cenc}_3(1, \mathsf{enc}(1, b_w)),$$
$$K_{w,3} = \mathsf{cenc}_3(j, \mathsf{enc}(j, r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}))$$

  If $f_w(x^*) = 0$ then there are three cases to consider. Let us first consider the case that $f_{A(w)}(x^*) = 0$, but $f_{B(w)}(x^*) = 1$ then $\mathcal{B}$ implicitly sets $a_w = c_{j+1} + \psi_w, b_w = \phi_w$ and $r_w = c_1 \cdot c_2 \cdots c_{j+1} + \eta_w$, where ψ_w, ϕ_w, η_w are chosen randomly using $\mathsf{samp}()$. Then the algorithm creates key components:

$$K_{w,1} = \mathsf{cenc}_3(1, \widetilde{c_{j+1}} + \mathsf{enc}(1, \psi_w)),\ K_{w,2} = \mathsf{cenc}_3(1, \mathsf{enc}(1, \phi_w)),$$

$$K_{w,3} = \mathsf{cenc}_3(j, \mathsf{enc}(j, \eta_w) - \psi_w \cdot \tilde{c_1} \cdots \tilde{c_j} - (\mathbf{y}^{j-1} \cdot \widetilde{c_{j+1}} + \mathsf{enc}(j, \psi_w)) \cdot \eta_{A(w)} - \mathsf{enc}(j, \phi_w \cdot r_{B(w)})$$

  For intuition regarding the above calculation, please refer to the previous proof.

  The case where $f_{B(w)}(x^*) = 0$ and $f_{A(w)}(x^*) = 1$ is performed in a symmetric manner to what is above, with the roles of a_w and b_w reversed.

  If both $f_{B(w)}(x^*) = 0$ and $f_{A(w)}(x^*) = 0$, then the computation is done exactly as above, except the computation of $K_{w,3}$ is as follows:

$$K_{w,3} = \mathsf{cenc}_3(j, \mathsf{enc}(j, \eta_w) - \psi_w \cdot \tilde{c_1} \cdots \tilde{c_j} - (\mathbf{y}^{j-1} \cdot \widetilde{c_{j+1}} + \mathsf{enc}(j, \psi_w)) \cdot \eta_{A(w)} - \phi_w \cdot (\tilde{c_1} \cdots \tilde{c_j} + \mathsf{enc}(j, \eta_{B(w)})))$$

  Remark. Again, note that these keys are distributed within negligible statistical distance to honestly generated keys due to rerandomization and the random choices of η_w, ψ_w, and ϕ_w.

For the output gate we chose η_w at random, where $w = n + q$. Thus, at the end we have $r_w = \prod_{i \in [1,k]} c_i + \eta_w$ for the output gate. This gives us a final cancellation in computing the "header" component of the key K_H, which should be a level $k-1$ encoding of $\alpha - r_w = \xi - \eta_w$. Thus, we can compute $K_H = \mathsf{cenc}_3(k-1, \xi - \eta_w)$. Note that this is distributed identically to the real distribution of K_H.

Guess. $\mathcal{B}$ receives back the guess $M' \in \{0, 1\}$ of the message from $\mathcal{A}$. If $M' = 1$ it guesses that T is a tuple; otherwise, it guesses that it is random.

This immediately shows that any adversary with non-trivial advantage in the KP-ABE selective security game will have an identical advantage in breaking the GGH-analog of the k-MDDH assumption. □