Identity-based Encryption from the Diffie-Hellman Assumption

NICO DÖTTLING, Helmholtz Center for Information Security (CISPA)
SANJAM GARG, University of California, Berkeley

We provide the first constructions of identity-based encryption and hierarchical identity-based encryption based on the hardness of the (Computational) Diffie-Hellman Problem (without use of groups with pairings) or Factoring. Our construction achieves the standard notion of identity-based encryption as considered by Boneh and Franklin [CRYPTO 2001]. We bypass known impossibility results using garbled circuits that make a non-black-box use of the underlying cryptographic primitives.

CCS Concepts: • **Security and privacy → Public key encryption**;

Additional Key Words and Phrases: Identity-based encryption, Garbled circuits

ACM Reference format:
Nico Döttling and Sanjam Garg. 2021. Identity-based Encryption from the Diffie-Hellman Assumption. *J. ACM* 68, 3, Article 14 (March 2021), 46 pages.
https://doi.org/10.1145/3422370

1 INTRODUCTION

Soon after the invention of public-key encryption [Diffie and Hellman 1976; Rivest et al. 1978], Shamir [Shamir 1984] posed the problem of constructing a public-key encryption scheme where encryption can be performed using just the *identity* of the recipient. In such an identity-based encryption (IBE) scheme there are four algorithms: (1) Setup generates the global public parameters and a master secret key, (2) KeyGen uses the master secret key to generate a secret key for the user with a particular identity, (3) Encrypt allows for encrypting messages corresponding to an identity, and (4) Decrypt can be used to decrypt the generated ciphertext using a secret key for the matching identity.

The ability of IBE to "compress" exponentially many public keys into "small" global public parameters [Boneh and Franklin 2001; Cocks 2001] provides a way for simplifying certificate management in e-mail systems. Specifically, Alice can send an encrypted email to Bob at `bob@iacr.org` by just using the string "`bob@iacr.org`" and the public parameters generated by a setup authority.

Research supported in part from AFOSR Award FA9550-19-1-0200, AFOSR YIP Award, AFOSR Award FA9550-15-1-0274, DARPA/ARL SAFEWARE Award W911NF15C0210, NSF SaTC Award 1936826, NSF CRII Award 1464397, and research grants by the Sloan Foundation, the Hellman Foundation and the Okawa Foundation, Visa Inc., and the Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the author and do not reflect the official policy or position of the funding agencies. Nico Döttling was supported by a post doc fellowship of the German Academic Exchange Service (DAAD).
Authors' addresses: N. Döttling, CISPA Campus, Stuhlsatzenhaus 5, 66123 Saarbrücken, Germany; email: nico.doettling@gmail.com; S. Garg, 685 Soda Hall, University of California, Berkeley 94720; email: sanjamg@berkeley.edu.

0004-5411/2021/03-ART14
https://doi.org/10.1145/3422370

In this solution, there is no need for Alice to obtain Bob's public key. Bob could decrypt the email using a secret key corresponding to "bob@iacr.org" that he can obtain from the setup authority or what is sometimes referred to as the private-key generator.[1]

The more functional notion of hierarchical IBE (HIBE) [Gentry and Silverberg 2002; Horwitz and Lynn 2002] additionally allows a user with a secret key for an identity id to generate a secret key for any identity $\mathsf{id}\|\mathsf{id}'$. For instance, in the example above, Bob can use the secret key corresponding to identity "bob@iacr.org" to obtain a secret key corresponding to the identity "bob@iacr.org||2017". Bob could then give this key to his secretary who could now decrypt all his emails tagged as being sent during the year 2017, while Bob is on vacation. HIBE has also proven to be a valuable tool in the construction of public key encryption schemes achieving stronger security, such as *forward secure public key encryption* [Canetti et al. 2003].

Security Models for IBE. The standard security notion of IBE is *full security* [Boneh and Franklin 2001]. This notion captures the idea that even an adversary that is allowed to see arbitrarily many user secret keys will not be able to learn anything about the message encrypted under an identity for which he has not learned the corresponding secret key. More specifically, this notion models security as a distinguishing experiment between a challenger and an adversary $\mathcal{A}$, both of which are PPT algorithms. The challenger runs the following experiment with the adversary. It first generates a pair $(\mathsf{mpk}, \mathsf{msk})$ of master public and secret keys and provides the mpk to the adversary $\mathcal{A}$. The adversary is further given access to a key-generation oracle that takes as inputs identities ID and outputs identity secret keys $\mathsf{sk}_{\mathsf{ID}}$ that are generated using the master secret key msk. At some point $\mathcal{A}$ outputs a challenge identity ID^* and two challenge messages m_0 and m_1 to the challenger. The challenger chooses a uniformly random bit b and provides an encryption c^* of m_b under the challenge identity ID^* to the adversary. After receiving c^*, $\mathcal{A}$ continues its computation with access to its key generation oracle. When it terminates, $\mathcal{A}$ outputs a guess b' for the challenge bit b. The adversary $\mathcal{A}$ wins the experiment if it has not queried an identity secret key for the challenge identity ID^* and its guess b' is correct, i.e., it holds that $b' = b$. An IBE scheme is called *fully secure* if no PPT adversary $\mathcal{A}$ wins this experiment with probability substantially better than 1/2, which effectively amounts to blind guessing. Note that in this experiment the adversary is allowed to choose the challenge identity ID^* *adaptively*, depending on the both the master public key mpk and the outputs of the key-generation oracle.

A weaker security notion for IBE, *selective security* [Boneh and Boyen 2004a; Canetti et al. 2003, 2004] is obtained by modifying the above experiment such that the adversary $\mathcal{A}$ needs to announce the challenge identity ID^* *before* seeing the master public key and getting access to the key-generation oracle. This notion amounts to modeling attacks against an *a priori* fixed static identity. The security definitions for HIBE are defined analogously, replacing the IBE key-generation oracle with a HIBE key-generation oracle and imposing the additional restriction that $\mathcal{A}$ is not allowed to query the key-generation oracle on any prefix of the challenge identity ID^*.

Constructions of IBE. The first IBE schemes were realized by Boneh and Franklin [2001] and Cocks [2001]. The construction of Boneh and Franklin makes use of a hard *bilinear group* [Joux 2000]. A bilinear group $\mathbb{G}$ is equipped with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, where the *target group* $\mathbb{G}_T$ is isomorphic to $\mathbb{G}$ but has a potentially different representation. The bilinear map $e(\cdot, \cdot)$ allows to compute the product of two values a and b hidden in group elements g^a and g^b by $e(g^a, g^b) = e(g, g)^{a \cdot b}$. Bilinear maps are the critical ingredient in the Boneh-Franklin construction.

[1] Of course, Bob needs to authenticate his identity to the setup authority in this step. Bob can perform this authentication in the same way as it would do to a Certification Authority while using a public-key encryption scheme.

On a high level, assume that an identity ID can be mapped into a group element g_{ID} and that the master public key is of the form $h = g^x$, where x is the master secret key. A user secret key corresponding to the identity ID is computed as $h_{\mathsf{ID}} = g_{\mathsf{ID}}^x$

To encrypt, the sender chooses a random value r and computes g^r and $e(g_{\mathsf{ID}}, h)^r$. Under a suitable assumption, namely, the bilinear Diffie-Hellman assumption [Joux 2000], the term $z = e(g_{\mathsf{ID}}, h)^r = e(g_{\mathsf{ID}}, g)^{xr}$ is pseudorandom given h, g_{ID} and g^r and can be used to blind a message. The critical feature of the bilinear map is that the receiver can recompute the same term z given g^r and h_{ID} by computing

$$e(h_{\mathsf{ID}}, g^r) = e(g_{\mathsf{ID}}^x, g^r) = e(g_{\mathsf{ID}}, g)^{xr}.$$

Boneh and Franklin [2001] propose to use a group equipped with a Weil-pairing [Joux 2000] as the bilinear map $e(\cdot, \cdot)$. For this instantiation, the group $\mathbb{G}$ is an elliptic curve group over a finite field $\mathbb{F}_p$ whereas the target group $\mathbb{G}_T$ is a subgroup of the multiplicative group $\mathbb{F}_p^*$. Breaking this scheme is no harder than computing discrete logarithms in $\mathbb{G}_T$, for which sub-exponential algorithms are known (cf. Section 3.2). In contrast, for elliptic curve groups without bilinear map, the currently best known algorithms to compute discrete logarithms have exponential runtime and it is conjectured that the problem is exponentially hard for such groups.

The construction of Cocks [2001] is based on the (by then) more standard quadratic residuosity assumption [Goldwasser and Micali 1982], which states that distinguishing squares modulo a Blum integer[2] N from random elements modulo N is hard. At the core of Cocks construction is a public key encryption scheme where the public key is a random quadratic residue a mod N, whereas the corresponding secret key is a square root r of a. Given an appropriate method to map identities to random quadratic residues, this public key encryption scheme gives rise to an IBE scheme. The master public key is now the modulus N, whereas the master secret key is the factorization of N. For a user public key a_{ID}, a corresponding user secret key r_{ID} such that $a_{\mathsf{ID}} = r_{\mathsf{ID}}^2 \bmod N$ can be efficiently computed given the factorization of N.

We remark that both the above-mentioned constructions were proven fully secure in the random oracle model [Bellare and Rogaway 1993; Canetti et al. 1998]. In this model, a hash function is modelled as a truly random function, and adversaries can only evaluate it via oracle access. Once a random oracle is instantiated with a concrete hash function, security proofs in this model lose their validity and have only *heuristic* quality. It took a considerable research effort to construct standard model schemes achieving full security in the bilinear setting [Boneh and Boyen 2004a, 2004b; Canetti et al. 2003; Okamoto and Takashima 2010; Waters 2005], whereas based on the quadratic residuosity assumption only selective security could be achieved in the standard model [Boneh et al. 2007]. Moreover, a long line of works led to fully secure HIBE in the standard model from assumptions in bilinear groups [Boneh and Boyen 2004a; Boneh et al. 2005; Gentry and Halevi 2009; Gentry and Silverberg 2002; Horwitz and Lynn 2002; Lewko and Waters 2010; Shi and Waters 2008; Waters 2009].

A different line of work, initiated by Gentry, Peikert, and Vaikuntanathan [Gentry et al. 2008] provided constructions of IBE from the Learning with Errors (LWE) assumption [Regev 2005]. The decisional LWE problem is defined as follows: Fix a modulus q. Given a matrix $\mathsf{A} \in \mathbb{Z}_q^{m \times n}$ and a vector $\mathsf{y} \in \mathbb{Z}_q^m$, distinguish whether y is uniformly random or of the form $\mathsf{A} \cdot \mathsf{s} + \mathsf{e}$, where $\mathsf{s} \in \mathbb{Z}_q^n$ is uniformly random and e is drawn from a suitable *short* distribution. A particularly desirable feature of LWE is that it enjoys *worst-case hardness*. Specifically, this means that provided that some underlying lattice problem is worst-case hard, then LWE is guaranteed to be hard on average.

[2] A Blum integer is a number of the form $N = P \cdot Q$ where P and Q are primes with $P = 2p + 1$ and $Q = 2q + 1$, such that p and q are also prime.

Moreover, the LWE assumption is conjectured to be intractable for efficient quantum algorithms, whereas essentially all number theoretic problems used in cryptography, which includes the assumptions used in this work, can be efficiently solved by variants of the Shor's algorithm [Shor 1994].

The construction of Gentry et al. [2008] is somewhat similar in spirit to the construction of Boneh and Franklin [2001]. The master public key is a random matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, whereas the master secret key is a *lattice trapdoor*, which for any given vector $\mathbf{a} \in \mathbb{Z}_q^n$ allows to sample *short* vectors $\mathbf{x}$ such that $\mathbf{a}^\top = \mathbf{x}^\top \cdot \mathbf{A}$. This is how identity secret keys are generated: First the identity is mapped to a vector $\mathbf{a}_{\mathsf{ID}}$ using a hash function (which is modelled as a random oracle), then a short vector $\mathbf{x}_{\mathsf{ID}}$ with $\mathbf{x}_{\mathsf{ID}}^\top \cdot \mathbf{A} = \mathbf{a}_{\mathsf{ID}}^\top$ is computed. To encrypt under an identity ID, a random vector $\mathbf{s}$ and short $\mathbf{e}$ and e are chosen and we compute $\mathbf{y} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ and $z = \mathbf{a}_{\mathsf{ID}}^\top \cdot \mathbf{s} + e$. Under the LWE assumption both $\mathbf{y}$ and z are pseudorandom, and security of the scheme can be established from this fact. To decrypt, note that given $\mathbf{y}$ and $\mathbf{x}_{\mathsf{ID}}$, we can compute

$$\begin{aligned}
\mathbf{x}_{\mathsf{ID}}^\top \cdot \mathbf{y} &= \mathbf{x}_{\mathsf{ID}}^\top \cdot (\mathbf{A} \cdot \mathbf{s} + \mathbf{e}) \\
&= \mathbf{x}_{\mathsf{ID}}^\top \cdot \mathbf{A} \cdot \mathbf{s} + \mathbf{x}_{\mathsf{ID}}^\top \cdot \mathbf{e} \\
&= \mathbf{a}_{\mathsf{ID}}^\top \cdot \mathbf{s} + e - e + \mathbf{x}_{\mathsf{ID}}^\top \cdot \mathbf{e} \\
&= z - e + \mathbf{x}_{\mathsf{ID}}^\top \cdot \mathbf{e} \\
&\approx z.
\end{aligned}$$

Thus, $\mathbf{x}_{\mathsf{ID}}^\top \cdot \mathbf{y}$ is close to z, which suffices to construct a correct IBE scheme using standard error correction techniques.

We note that this construction critically relies on the *lattice trapdoor* in very much the same way as the Boneh-Franklin construction relies on the bilinear pairing. However, public key encryption schemes can be constructed from LWE without the use of a lattice trapdoor [Regev 2005]. Yet, unlike in the group-based setting where using a bilinear map necessitates the use of special bilinear groups and therefore makes a strong qualitative difference, using lattice trapdoors in the LWE setting makes more of a quantitative difference. Particularly an LWE assumption with slightly stronger parameters, that is a slightly stronger modulus-to-noise ratio is needed. Subsequent LWE-based constructions achieved selectively secure HIBE and removed the need for random oracles [Agrawal et al. 2010a, 2010b; Agrawal and Boyen 2009; Cash et al. 2010]. We remark that achieving fully secure HIBE is generally considered a hard problem. To date, the only construction achieving this notion from a standard assumption is based on bilinear maps [Waters 2009].

We observe that all the constructions of IBE discussed above follow the same basic blueprint. First a public key encryption scheme is constructed for which public keys, relative to some master public key, can be chosen *obliviously*, i.e., without knowledge of a corresponding secret key. Then, a mechanism is provided that, given a special trapdoor, allows to compute secret keys for any given public key *after the fact*. For a given identity, encryption proceeds in two steps, first deriving a public key corresponding to the identity, and then encrypting the message under this public key. Such a special trapdoor is not necessary to achieve public key encryption, and this makes achieving IBE a seemingly harder task than just achieving public key encryption. Moreover, it seems hard to construct IBE for assumptions that do not support such a special trapdoor.

Practical Aspects of IBE. We remark that using IBE as a form of identity management in a large-scale network is potentially dangerous because of the *key-escrow* problem. Specifically, the use of a master secret key introduces a single point of failure into the entire network, as possession of this key allows to decrypt any ciphertext for any identity relative to the master public key. Fur-

thermore, exfiltrating the master secret key constitutes a high value target for an attacker seeking to compromise the security of such a network. Consequently, for large networks measures would need to be put in place to prevent such an attack. For example, with the goal of making the trust de-centralized, Boneh and Franklin [2001] suggested the use of multiple setup authorities, instead of just one. Very recently, building crucially on the techniques developed in this article, a new variant of IBE that allows each user to register a public key corresponding to their identity have been developed [Garg et al. 2018, 2019]. This allows for removing the need of key-escrow completely. However, these new schemes need updates to public parameters as new users register with their public keys. Additionally, users also need to obtain infrequent updates to the identity specific decryption keys.

Finally, IBE has found numerous applications beyond its original use case. IBE can be used as a versatile access control mechanism, both in theoretical and practical applications. For instance, IBE and HIBE can be used to delegate reading rights in an encrypted file system (see the discussion in Section 3.7). In this example, the *users* obtaining identity secret keys would be programs requesting read access to encrypted files, where, e.g., the filename constitutes the identity of an encrypted file. In terms of theoretical applications, IBE has been used to construct chosen-ciphertext secure public key encryption [Canetti et al. 2004] and garbled RAM [Gentry et al. 2014].

1.1 Our Results

In this work, we show a fully secure construction of IBE and a selectively secure HIBE based just on the Computational Diffie-Hellman (CDH) assumption or what we sometimes refer to as the Diffie-Hellman assumption. In the group of quadratic residues this problem is as hard as the Factoring problem [Biham et al. 1997; McCurley 1988; Shmuely 1985]. Therefore, this implies a solution based on the hardness of factoring as well. We note that our results are in the standard model, i.e., they do not rely on the random oracle heuristic [Bellare and Rogaway 1993; Canetti et al. 1998].

The CDH assumption for a cyclic group $\mathbb{G}$ of prime order p states that it is hard to compute $g^{a \cdot b}$ given only g, g^a, g^b for random $a, b \xleftarrow{\$} \mathbb{Z}_p$. The CDH assumption is, in some sense, the weakest assumption in cyclic groups known to imply public key encryption. In other words, any assumption in cyclic groups based on which constructions of public key encryption are known implies the hardness of the CDH assumption. This naturally includes all assumptions on bilinear groups, such as the bilinear Diffie Hellman assumption [Joux 2000] or the d-LIN assumption [Hofheinz and Kiltz 2007; Waters 2009]. In this sense, our work provides a construction of IBE from the minimal assumption needed to even achieve public key encryption in the group setting. As mentioned earlier, for all known bilinear groups subexponential attacks are known. However, there are candidate groups in which the best known-algorithms for the CDH problem have exponential runtime. Moreover, by our current understanding the factoring assumption is strictly weaker than the quadratic residuosity assumption used in Cocks [2001] and Boneh et al. [2007], i.e., while the quadratic residuosity assumption immediately implies the factoring assumption, the converse is not known. The LWE assumption, however is, by our current understanding, incomparable to both the CDH and the factoring assumption.

Our constructions bypass known impossibility results [Boneh et al. 2008; Papakonstantinou et al. 2012] by making a non-black-box use of the underlying cryptographic primitives. However, this non-black-box use of cryptographic primitives also makes our scheme relatively inefficient. In Section 6, we suggest ideas for reducing the non-black-box use of the underlying primitives, thereby improving the efficiency of our scheme. Even with these optimizations, our IBE scheme

is prohibitive when compared with the IBE schemes based on bilinear maps. We leave open the problem of realizing an efficient IBE scheme from the CDH assumption.

Finally, we remark that for our constructions we do not need to make additional requirements on the underlying group. For instance, we do not require that the underlying group has a dense representation. Recall that in the Boneh-Franklin construction, it is critical that group elements can be generated obliviously so as to allow to map identities to group elements via hashing.

Subsequent work. In a followup work [Döttling and Garg 2017], we show how the techniques from this article can be used to obtain generic constructions of fully secure IBE and selectively secure HIBE starting with any selectively secure IBE scheme. Another follow-up work [Döttling et al. 2018] obtains a construction of IBE based on learning parity with noise (LPN). Furthermore, it improves on the parameters when constructing schemes from learning with errors (LWE). Specifically, using techniques developed in this work they overcome the need of a *lattice trapdoor*. This yields an IBE construction from LWE with the same parameters that are currently needed to even achieve public key encryption [Regev 2005].

In a wider perspective, the techniques developed in this work allow constructing IBE from weaker assumptions with less structure. Yet, this is currently achieved at the expense of *efficiency*.

1.2 Black-box Impossibility of IBE in the Generic Group Model

Before providing an overview of the results of this work, we will discuss the so-called *black-box barriers* for IBE. Black-box constructions are oblivious of the way the base primitive is implemented. More specifically, the algorithms of the target primitive only have oracle access to the algorithms of the base primitive.[3] The first black-box impossibility result for IBE is due to Boneh, Papakonstantinou, Rackoff, Vahlis, and Waters [Boneh et al. 2008]. They showed that IBE cannot be realized using trapdoor permutations or CCA-secure public-key encryption in a black-box manner. This result is established by showing that any IBE construction making only black-box use of trapdoor permutations is insecure, i.e., there exists an attacker breaking the scheme. Extending this work, Papakonstantinou, Rackoff, and Vahlis [Papakonstantinou et al. 2012] showed that black-box use of a group over which, e.g., the Computational Diffie-Hellman (CDH) problem or the Decisional Diffie-Hellman (DDH) problem are assumed to be hard is also insufficient for realizing IBE.

Since the result of Papakonstantinou et al. [2012] builds upon ideas of Boneh et al. [2008], we will first outline the ideas behind Boneh et al. [2008]. For the sake of argument, assume that a black-box construction of IBE from trapdoor permutations (TDPs) exists. The main idea in Boneh et al. [2008] is that IBE achieves a form of *key compression*, in that exponentially many public keys corresponding to the identities are succinctly represented in the master public key mpk. Furthermore, this compression must be achieved while keeping the size of the master public key small (i.e., polynomial in the security parameter). Therefore, mpk can only encode a small set K of TDP public keys. Moreover, an *attacker* can identify this set of TDP public keys with matching trapdoors as follows: Given a master public key mpk, the attacker efficiently obtains the set of TDP public keys in K along with the corresponding trapdoors (or at least the key trapdoor pairs that are most frequently used) by encrypting random messages under many random identities and then decrypting the generated ciphertexts with user secret keys (obtained from the key generation oracle) for the used identities. Throughout this process the attacker keeps track of all the TDP

[3]Constructing sophisticated cryptographic primitives from simpler primitives in a black-box fashion is a well-established goal in cryptography. Black-box constructions are highly desired, as they tend to be more efficient than the non-black-box ones. This inefficiency stems mainly from the fact that non-black-box techniques typically unroll the base primitive as a Boolean circuit and add another layer of cryptographic operations around it.

public keys used by encryption in its oracle queries. Additionally, the attacker tracks the trapdoors used in the TDP inversion queries.[4] Again, this can be achieved as encryption only makes black-box use of the TDP. Next, Boneh et al. [2008] argue that this critical set of TDP public keys K along with the corresponding trapdoors effectively include all the secret information in the master secret key and can be used to decrypt any ciphertext. In more detail, to complete the attack, the attacker resamples an approximation msk^* of msk consistent with the TDP public keys and corresponding trapdoors recovered previously. Next, it chooses a random challenge identity ID^* and obtains a challenge ciphertext c^* under ID^*. Finally, it uses msk^* to compute an identity secret key $\mathsf{sk}^*_{\mathsf{ID}^*}$ and uses it to decrypt c^*.

Papakonstantinou et al. [2012] extend this impossibility result to constructions that are based on hardness assumptions in cyclic groups and only make black-box use of the underlying group. Specifically, they prove their results in the *generic group model* [Shoup 1997], where both the primitive and the adversary have only black-box access to the group. In this model, all group operations are performed by an oracle, which only outputs *handles* to group elements. Such handles reveal nothing about the actual representation of group elements. Consequently, in the generic group model essentially all standard assumptions in cyclic groups such as the discrete logarithm (DL), CDH, and DDH assumptions and many more are *provably* hard.

The basic idea behind the attack of Papakonstantinou et al. [2012] is similar to the idea of Boneh et al. [2008], with the difference that instead of considering TDP public keys encoded in the master public key mpk, we now consider group elements $\mathsf{h}_1, \ldots, \mathsf{h}_\ell$ encoded in mpk and the secret key encodes discrete logs of these group elements or linear relations between the discrete logs of these group elements. Here h_i is of the form $\mathsf{h}_i = \mathsf{g}^{x_i}$ for a public generator g and a secret exponent x_i. The basic idea of the attack is that every time the adversary $\mathcal{A}$ queries a user secret key, it learns a new *linear relation* between the x_i.

Analogous to Boneh et al. [2008], Papakonstantinou et al. [2012] show that an attacker can *discover* the group elements encoded in the public key along with linear relations in their discrete logs as follows: Specifically, the attacker obtains these values by encrypting random messages under many random identities and then decrypting the generated ciphertexts with user secret keys (obtained from the key generation oracle) for the used identities. Sufficient repetition of this process will essentially reveal all the linear relations necessary for performing decryption for any identity.

To complete the attack, the attacker resamples an approximation msk^* of msk consistent with available information. Next, it chooses a random challenge identity ID^* and obtains a challenge ciphertext c^* under ID^*. Finally, it uses msk^* to compute an identity secret key $\mathsf{sk}^*_{\mathsf{ID}^*}$ and uses it to decrypt c^*.

1.3 Overcoming the Impossibility: Using Garbled Circuits

In the previous sub-section, we discussed how the black-box use of a group was insufficient for realizing IBE. Thus, any construction of IBE from assumptions in cyclic groups must necessarily make non-black-box use of the group. We will now provide a high-level discussion of how our work overcomes this barrier using garbled circuits. We provide details in Section 2.

Garbled circuits [Bellare et al. 2012; Lindell and Pinkas 2009; Yao 1982] are a cryptographic tool for securely computing arbitrary circuits. Specifically, a garbling scheme consists of two algorithms, a garbling algorithm GCircuit and an evaluation algorithm Eval. The algorithm GCircuit

[4] A cryptographic scheme, making black-box use of an underlying primitive, cannot hide the inputs on which the underlying primitive is executed. Thus, during the execution of any algorithms of the scheme, the executing party can observe the inputs that are fed into the underlying primitive and resulting the outputs.

takes as input a circuit C and outputs a garbled circuit $\widetilde{\mathsf{C}}$ and a set of *keys* or labels $(\mathsf{key}_{i,b})_{i,b}$. Via the Eval procedure, a garbled circuit $\widetilde{\mathsf{C}}$ can be evaluated on an input x given the keys $(\mathsf{key}_{i,x_i})_i$ *encoding* to the input x, resulting in $\mathsf{C}(x)$. Thus, computation of C on an input x can be delegated to an untrusted worker by providing the keys corresponding to x to the worker. Security of garbling schemes is defined via a simulation-based notion. Specifically, we require the existence of a simulator Sim that takes as input y and outputs a garbled circuit $\widetilde{\mathsf{C}}$ and keys $(\mathsf{key}_i)_{i\in[n]}$ such that for any input x

$$(\widetilde{\mathsf{C}}, (\mathsf{key}_{i,x_i})_i) \overset{c}{\approx} \mathsf{Sim}(\mathsf{C}(x)),$$

where $(\widetilde{\mathsf{C}}, (\mathsf{key}_{i,b})_{i,b}) := \mathsf{GCircuit}(\mathsf{C}(x))$ and $\overset{c}{\approx}$ denotes computational indistinguishability. In other words, the garbled circuit $\widetilde{\mathsf{C}}$ together with the labels encoding x can be simulated given only $\mathsf{C}(x)$ but without any further knowledge of x.

In our scheme, ciphertexts contain a sequence of garbled circuits. The decryptor will be able to execute these garbled circuits on input group elements that are not contained in or accessible from the master public key. Since the underlying circuit performs group operations, this construction is no longer black-box in the group. A critical aspect in this context is that the encryptor does not know the input to the garbled circuit, but will nevertheless be able to communicate the input keys corresponding to appropriate group elements to the decryptor, while ensuring that each garbled circuit can only be executed on a unique input. Consequently, our encryption procedure will not be limited to using only the few group elements contained in the master public key, but will be able to use group elements that are inaccessible and unknown to the encryptor. This limitation was the core reason for the impossibility of Papakonstantinou et al. [2012].

Our construction consequently also deviates from the classical IBE construction paradigm discussed in the introduction. There is no *natural* public key encryption scheme associated with our construction such that IBE encryption consists in first deriving a (unique) user public key from the identity, and then using this key to encrypt the message. Instead, IBE encryption in our scheme will make a more sophisticated use of the identity-string.

2 OUR TECHNIQUES

In this section, we give an intuitive explanation of our construction of IBE from the DDH assumption. We defer the details on obtaining the same results based on CDH to the main body of the article. In a nutshell, the DDH assumption for a group $\mathbb{G}$ of order p states that the distributions $(g, g^u, g^v, g^{u\cdot v})$ and (g, g^u, g^v, g^r) are computationally indistinguishable, where $u, v, r \overset{\$}{\leftarrow} \mathbb{Z}_p$ and $g \overset{\$}{\leftarrow} \mathbb{G}$ are uniformly random. We start by describing a new primitive called *chameleon encryption scheme*—a chameleon hash function [Krawczyk and Rabin 1998] that additionally supports certain encryption and decryption procedures.[5] Subsequently, we describe how chameleon encryption along with garbled circuits can be used to realize IBE.

2.1 Chameleon Encryption

As mentioned above, a chameleon encryption scheme is a chameleon hash function that supports certain encryption and decryption procedures along with. We start by describing what this new primitive is and then describe how we construct it. Recall that a chameleon hash function is a collision resistant hash function for which the knowledge of a trapdoor enables collision finding.

[5]The notion of chameleon hashing is closely related to the notion of chameleon commitment scheme [Brassard et al. 1988] and we refer the reader to Krawczyk and Rabin [1998] for more discussion on this.

Our Chameleon Hash. Given a cyclic group $\mathbb{G}$ of prime order p with a generator g, consider the following chameleon hash function:

$$\mathsf{H}(\mathsf{k}, \mathsf{x}; r) = g^r \prod_{j \in [n]} g_{j,\mathsf{x}_j},$$

where $\mathsf{k} = (g, \{g_{j,0}, g_{j,1}\}_{j\in[n]})$, $r \in \mathbb{Z}_p$ and x_j is the jth bit of $\mathsf{x} \in \{0,1\}^n$. It is not very hard to see that this hash function is (i) collision-resistant based on the hardness of the discrete-log problem, and (ii) chameleon given the trapdoor information $\{\mathsf{dlog}_g\, g_{j,0}, \mathsf{dlog}_g\, g_{j,1}\}_{j\in[n]}$—specifically, given any $\mathsf{x}, r, \mathsf{x}'$ and the trapdoor information, we can efficiently compute r' such that $\mathsf{H}(\mathsf{k}, \mathsf{x}; r) = \mathsf{H}(\mathsf{k}, \mathsf{x}'; r')$.

The Associated Encryption—Abstractly. Corresponding to a chameleon hash function, we require encryption and decryption algorithms such that

(1) Encryption $\mathsf{Enc}(\mathsf{k}, (\mathsf{h}, i, b), \mathsf{m})$: On input a key k, a hash value h, a location $i \in [n]$, a bit $b \in \{0,1\}$, and a message $\mathsf{m} \in \{0,1\}$ outputs a ciphertext ct, and
(2) Decryption $\mathsf{Dec}(\mathsf{k}, (\mathsf{x}, r), \mathsf{ct})$: On input a ciphertext ct, a bit-string x and coins r yields m if

$$\mathsf{h} = \mathsf{H}(\mathsf{k}, \mathsf{x}; r) \text{ and } \mathsf{x}_i = b,$$

where (h, i, b) are the values used in the generation of the ciphertext ct.

In other words, the decryptor can use the knowledge of the preimage of h as the key to decrypt m as long as the ith bit of the preimage it can supply is equal to the value b chosen at the time of encryption. Our security requirement roughly states that

$$\{\mathsf{k}, \mathsf{x}, r, \mathsf{Enc}(\mathsf{k}, (\mathsf{h}, i, 1-\mathsf{x}_i), 0)\} \overset{c}{\approx} \{\mathsf{k}, \mathsf{x}, r, \mathsf{Enc}(\mathsf{k}, (\mathsf{h}, i, 1-\mathsf{x}_i), 1)\},$$

where $\overset{c}{\approx}$ denotes computational indistinguishability. That is, the encrypted message is hidden if the decryptor has the *wrong* preimage of h, i.e., a preimage x where the ith bit x_i is different from the bit b used by encryption.

The Associated Encryption—Realization. Corresponding to the chameleon hash defined above our encryption procedure $\mathsf{Enc}(\mathsf{k}, (\mathsf{h}, i, b), \mathsf{m})$ proceeds as follows: Sample a random value $\rho \overset{\$}{\leftarrow} \mathbb{Z}_p$ and output the ciphertext $\mathsf{ct} = (e, c, c', \{c_{j,0}, c_{j,1}\}_{j\in[n]\setminus\{i\}})$ where

$$\begin{aligned} & c := g^\rho && c' := \mathsf{h}^\rho, \\ \forall j \in [n]\setminus\{i\}, \quad & c_{j,0} := g_{j,0}^\rho && c_{j,1} := g_{j,1}^\rho, \\ & e := \mathsf{m} \oplus g_{i,b}^\rho. \end{aligned}$$

If $\mathsf{x}_i = b$, then decryption $\mathsf{Dec}(\mathsf{ct}, (\mathsf{x}, r))$ can just output

$$e \oplus \frac{c'}{c^r \prod_{j\in[n]\setminus\{i\}} c_{j,\mathsf{x}_j}}.$$

It can easily be seen that this expression is identical to the message m. However, if $\mathsf{x}_i \neq b$, then the decryptor has access to the value g_{i,x_i}^ρ but not $g_{i,b}^\rho$, and this prevents him from learning the message m. Formalizing this intuition, we can argue security of this scheme based on the DDH assumption.[6] In a bit more detail, we can use an adversary $\mathcal{A}$ breaking the security of the chameleon encryption

[6]In Section 5, we explain our constructions of chameleon encryption based on the CDH Assumption, or the Factoring Assumption.

scheme to distinguish DDH tuples (g, g^u, g^v, g^{uv}) from random tuples (g, g^u, g^v, g^s). Fix (adversarially chosen) $\mathsf{x} \in \{0,1\}^n$, index $i \in [n]$ and a bit $b \in \{0,1\}$. Given a tuple (g, U, V, T), we can simulate the public key k, hash value h, coins r, and ciphertext ct as follows: Choose uniformly random values $\alpha_{j,0}, \alpha_{j,1} \xleftarrow{\$} \mathbb{Z}_p$ and set $g_{j,0} = g^{\alpha_{j,0}}$ and $g_{j,1} = g^{\alpha_{j,1}}$ for $j \in [n]$. Now *reassign* $g_{i,1-\mathsf{x}_i} = U$ and set $\mathsf{k} := (g, \{g_{j,0}, g_{j,1}\}_{j\in[n]})$. Choose $r \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and set $\mathsf{h} := \mathsf{H}(\mathsf{k}, \mathsf{x}; r)$. Finally, prepare a challenge ciphertext $\mathsf{ct} := (e, c, c', \{c_{j,0}, c_{j,1}\}_{j\in[n]\setminus\{i\}})$ by choosing

$$
\begin{aligned}
& c := V && c' := V^r \cdot \prod_{j\in[n]} V^{\alpha_{j,\mathsf{x}_j}}, \\
\forall j \in [n]\setminus\{i\}, \quad & c_{j,0} := V^{\alpha_{j,0}} && c_{j,1} := V^{\alpha_{j,1}}, \\
& e := \mathsf{m} \oplus T,
\end{aligned}
$$

where $\mathsf{m} \in \{0,1\}$. Now, if $(g, U, V, T) = (g, g^u, g^v, g^{uv})$, then a routine calculation shows that k, h, r and ct have the proper distribution, thus $\mathcal{A}$'s advantage in guessing m remains the same as in the real experiment. However, if T is chosen uniformly at random and independent of g, U, V, then $\mathcal{A}$'s advantage to guess m given k, h, r, and ct is obviously 0, which concludes this proof-sketch.

Connection to Witness Encryption. A witness encryption (WE) scheme [Garg et al. 2013] (see also Ananth et al. [2013] and Boyle et al. [2014a]) for an NP language $\mathcal{L}$ is a type of public key encryption, which allows to encrypt messages under *statements* x. More precisely, encryption takes as input a statement x and a message m and outputs a ciphertext c. In a way, in this notion the statement x replaces the public key in standard public key encryption. Given such a ciphertext c for a statement $x \in \mathcal{L}$, any witness w for $x \in \mathcal{L}$ (under a pre-defined witness relation) can be used to decrypt the ciphertext c. However, if $x \notin \mathcal{L}$, then the ciphertext c computationally hides the encrypted message m. Witness encryption has been shown to imply IBE when combined with a suitable signature scheme [Garg et al. 2013].

When considered through the lens of witness encryption, Chameleon encryption can be seen as an *average-case* witness encryption scheme for a specific hash function. Given an "instance" represented by a key k, a hash value h, an index i, and a bit b, Chameleon encryption allows to encrypt a message m into a ciphertext c. Now, anyone in possession of a preimage x of h subject to $x_i = b$ (serving as the witness) will be able to decypt c.

While security of witness encryption is guaranteed for encryptions under any *no-instance*, this is not the case for chameleon encryption. In the case of chameleon encryption, the programmability property implies that there are no no-instances. That is, every input x can be mapped to every hash value h for an appropriate choice of r. Our security definition for Chameleon encryption avoids this issue by requiring security only to hold for randomly chosen keys k rather than worst-case instances.

2.2 From Chameleon Encryption to Identity-based Encryption

The public parameters of an IBE scheme need to encode exponentially many public keys succinctly—one per each identity. Subsequently, corresponding to these public parameters the setup authority should be able to provide the secret key for any of the exponentially many identities. This is in sharp contrast with public-key encryption schemes for which there is only one trapdoor per public key, which if revealed leaves no security. At a very high level, our chameleon hash function is the ingredient that allows for compressing exponentially many public keys into small public parameters.

Arrangement of the keys. We start by describing the arrangement of the exponentially many keys in our IBE scheme for identities of length n bits. First, imagine a fresh encryption-decryption

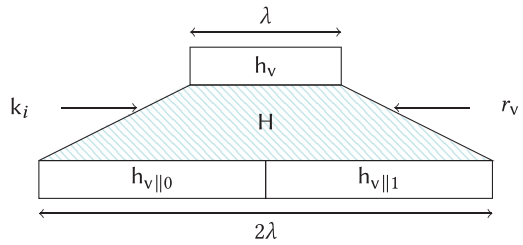

Fig. 1. The hashing relation for a node v in our IBE construction. We separate the inputs to $\mathsf{H}(\mathsf{k}_G, \cdot)$ into two blocks. When we compute hash encryptions, they will address one of the four blocks.

key pair for any public-key encryption scheme for each identity in $\{0,1\}^n$. We will denote this pair for identity $\mathsf{v} \in \{0,1\}^n$ by $(\mathsf{ek}_\mathsf{v}, \mathsf{dk}_\mathsf{v})$. Next, to set up the hash values, we sample n hash keys—namely, $\mathsf{k}_0, \ldots \mathsf{k}_{n-1}$. Now, consider a tree of depth n and for each node $\mathsf{v} \in \{0,1\}^{\leq n-1} \cup \{\epsilon\}$[7] the hash value h_v is set as

$$\mathsf{h}_\mathsf{v} = \begin{cases} \mathsf{H}(\mathsf{k}_i, \mathsf{ek}_{\mathsf{v}\|0} \| \mathsf{ek}_{\mathsf{v}\|1}; r_\mathsf{v}) & \mathsf{v} \in \{0,1\}^{n-1} \text{ where } i = |\mathsf{v}|, \\ \mathsf{H}(\mathsf{k}_i, \mathsf{h}_{\mathsf{v}\|0} \| \mathsf{h}_{\mathsf{v}\|1}; r_\mathsf{v}) & \mathsf{v} \in \{0,1\}^{<n-1} \cup \{\epsilon\} \text{ where } i = |\mathsf{v}|, \end{cases} \tag{1}$$

where for each $\mathsf{v} \in \{0,1\}^{<n} \cup \{\epsilon\}$ the coins r_v are chosen randomly. This arrangement of values is illustrated in Figure 1.

Generating the tree on demand. Note that the setup authority cannot generate and hash these exponentially many hash keys at setup time. Instead, it generates them implicitly. More specifically, the setup authority computes each h_v as $\mathsf{H}(\mathsf{k}_{|\mathsf{v}|}, 0^{2\lambda}; \omega_\mathsf{v})$. Then, later on when needed, using the trapdoor $\mathsf{t}_{|\mathsf{v}|}$ for the hash key $\mathsf{k}_{|\mathsf{v}|}$, we can obtain coins r_v such that the generated value h_v indeed satisfies Equation (1). Furthermore, to maintain consistency (in the tree and across different invocations) the randomness ω_v used for each v is chosen using a pseudorandom function. In summary, with this change the entire tree can be represented succinctly and generated on demand. Analogously, the public key and secret key pairs $(\mathsf{ek}_\mathsf{v}, \mathsf{dk}_\mathsf{v})$ for each identity $\mathsf{v} \in \{0,1\}^n$ are generated using a pseudorandom function.

What are the public parameters? Note that the root hash value h_ϵ in a way binds the entire tree of hash values. With this in mind, we set the public parameters of the scheme to be the n hash keys and the root hash value, i.e.,

$$\mathsf{k}_0, \ldots \mathsf{k}_{n-1}, \mathsf{h}_\epsilon.$$

Secret-key for a particular identity ID. Given the above tree structure, the secret key for some identity ID simply consists of the hash values along the path from the root to the leaf corresponding to ID and their siblings along with the decryption key dk_ID.[8] Specifically, the secret key sk_ID for identity ID consists of $(\{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v} \in V}, \mathsf{dk}_\mathsf{ID})$ where $V := \{\varepsilon, \mathsf{ID}[1], \ldots \mathsf{ID}[1 \ldots n-1]\}$ and

$$\mathsf{lk}_\mathsf{v} = \begin{cases} (\mathsf{h}_\mathsf{v}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|1}, r_\mathsf{v}) & \text{for } \mathsf{v} \in V \backslash \{\mathsf{ID}[1 \ldots n-1]\} \\ (\mathsf{h}_\mathsf{v}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1}, r_\mathsf{v}) & \text{for } \mathsf{v} = \mathsf{ID}[1 \ldots n-1] \end{cases}.$$

Encryption and Decryption. Before providing details of encryption and decryption, we will briefly discuss how chameleon encryption can be useful in conjunction with garbled circuits.[9] Chameleon encryption allows an encryptor knowing a key k and a hash value h to encrypt a set

[7] We use ϵ to denote the empty string.

[8] We note that our key generation mechanism can be seen as an instantiation of the Naor and Yung [1989] tree-based construction of signature schemes from universal one-way hash functions and one-time signatures. This connection becomes even more apparent in the followup work Döttling and Garg [2017].

[9] For this part of the intuition, we assume familiarity with garbled circuits.

of labels $\{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}_j$ such that a decryptor knowing x and r with $\mathsf{H}(\mathsf{k}, \mathsf{x}; r) = \mathsf{h}$ can recover $\{\mathsf{lab}_{j,\mathsf{x}_j}\}_j$. However, security of chameleon encryption guarantees that the receiver learns nothing about the remaining labels. In summary, using this mechanism, the generated ciphertexts enable the decryptor to feed x into a garbled circuit to be processed further.

To encrypt a message m to an identity $\mathsf{ID} \in \{0,1\}^n$, the encryptor will generate a sequence of $n+1$ garbled circuits $\{\tilde{P}^0, \ldots \tilde{P}^{n-1}, \tilde{T}\}$ such that a decryptor in possession of the identity secret key $\mathsf{sk}_{\mathsf{ID}} = (\{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v} \in V}, \mathsf{dk}_{\mathsf{ID}})$ will be able to evaluate these garbled circuits one after another. Roughly speaking, circuit P^i for any $i \in \{0 \ldots n-1\}$ and $\mathsf{v} = \mathsf{ID}[1 \ldots i]$ takes as input a hash value h_v and generates chameleon encryptions of the input labels of the next garbled circuit $\tilde{P}^{i+1}$ using a key $\mathsf{k}_{|\mathsf{v}|}$ hardwired inside it and the hash value h given to it as input (in a manner as described above). The last circuit T will just take as input an encryption key $\mathsf{ek}_{\mathsf{ID}}$ and output an encryption of the plaintext message m under $\mathsf{ek}_{\mathsf{ID}}$. Finally, the encryptor provides input labels for the first garbled circuit $\tilde{P}^0$ for the input h_ε in the ciphertext.

During decryption, for each $i \in \{0 \ldots n-1\}$ and $\mathsf{v} = \mathsf{ID}[1 \ldots i]$ the decryptor will use the local key lk_v to decrypt the ciphertexts generated by $\tilde{P}^i$ and obtain the input labels for the garbled circuits $\tilde{P}^{i+1}$ (or, T if $i = n-1$). We will now explain the first iteration of this construction in more detail. All further iterations proceed analogously. The encryptor provides garbled input labels corresponding to input h_ε for the first garbled circuit $\tilde{P}^0$ in the ciphertext. The garbled circuit $\tilde{P}^0$ has $\mathsf{ID}[1]$ and the input labels $\{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}_{j \in [\lambda]}$ hardwired in it. Thus, the decryptor can evaluate $\tilde{P}^0$ and obtain encryptions of input labels $\{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}_{j \in [\lambda]}$ for the circuit $\tilde{P}^1$, namely:

$$\left\{\mathsf{Enc}(\mathsf{k}_0, (\mathsf{h}_\varepsilon, \mathsf{ID}[1] \cdot \lambda + j, 0), \mathsf{lab}_{j,0}), \qquad \mathsf{Enc}(\mathsf{k}_0, (\mathsf{h}_\varepsilon, \mathsf{ID}[1] \cdot \lambda + j, 1), \mathsf{lab}_{j,1})\right\}_{j \in [\lambda]}.$$

Given these encryptions, the decryptor uses $\mathsf{lk}_\varepsilon = (\mathsf{h}_\varepsilon, \mathsf{h}_0, \mathsf{h}_1, r_\varepsilon)$ to learn the garbled input labels $\{\mathsf{lab}_{j,\mathsf{h}_{\mathsf{ID}[1]},j}\}_{j \in [\lambda]}$ where $\mathsf{h}_{\mathsf{ID}[1],j}$ is the jth bit of $\mathsf{h}_{\mathsf{ID}[1]}$. In other words, the decryptor now possesses input labels for the input $\mathsf{h}_{\mathsf{ID}[1]}$ for the garbled circuit $\tilde{P}^1$ and can therefore evaluate $\tilde{P}^1$. Analogous to the previous step, the decryptor uses $\mathsf{lk}_{\mathsf{ID}[1..2]}$ to obtain input labels to $\tilde{P}^2$ and so on. The decryptor's ability to provide the local keys lk_v for $v \in V$ keeps this process going, ultimately evaluating $\tilde{\mathsf{T}}$, which reveals an encryption of the message m under the encryption key $\mathsf{ek}_{\mathsf{ID}}$. This final ciphertext can be decrypted using the decryption key $\mathsf{dk}_{\mathsf{ID}}$. At a high level, our encryption method (and the use of garbled circuits for it) has similarities with garbled RAM schemes [Garg et al. 2015a, 2015b; Gentry et al. 2014; Lu and Ostrovsky 2013] and relates most closely with Cho et al. [2017]. Full details of the construction are provided in Section 6.

Proof Sketch. The intuition behind the proof of security, which follows by a sequence of hybrid changes, is as follows: The first (easy) change is to replace the pseudorandom function used to generate the local keys by a truly random function, something that should go undetected against a computationally bounded attacker. Next, via a sequence of hybrids, we change the $n+1$ garbled circuits $\tilde{P}^0, \ldots \tilde{P}^{n-1}, \tilde{T}$ to their simulated versions one by one. Once these changes are made, the simulated circuit $\tilde{T}$ just outputs an encryption of the message m under the encryption key $\mathsf{ek}_{\mathsf{ID}^*}$ corresponding challenge identity ID^*, which hides m based on semantic security of the encryption scheme.

The only "tricky" part of the proof is the one that involves changing garbled circuits to their simulated versions. In this intuitive description, we explain how the first garbled circuit $\tilde{P}^0$ is moved to its simulated version. The argument for the rest of the garbled circuits is analogous. This change involves a sequence of four hybrid changes.

(1) First, we change how h_ε is generated. As a quick recap, recall that h_ε is generated as $\mathsf{H}(\mathsf{k}_0, 0^{2\lambda}; \omega_\varepsilon)$ and r_ε are set to $\mathsf{H}^{-1}(\mathsf{t}_0, (0^{2\lambda}, \omega_\varepsilon), \mathsf{h}_0 \| \mathsf{h}_1)$, where $\mathsf{H}^{-1}(\mathsf{t}_0, (0^{2\lambda}, \omega_\varepsilon), \mathsf{h}_0 \| \mathsf{h}_1)$

uses the collision-trapdoor t_0 to compute coins r_ε such that $\mathsf{H}(\mathsf{k}_0, 0^{2\lambda}; \omega_\varepsilon) = \mathsf{H}(\mathsf{k}_0, \mathsf{h}_0 \| \mathsf{h}_1; r_\varepsilon)$. We instead generate h_ε directly to be equal to $\mathsf{H}(\mathsf{k}_0, \mathsf{h}_0 \| \mathsf{h}_1, r_\varepsilon)$ using fresh coins r_ε. The trapdoor collision and uniformity properties of the chameleon encryption scheme ensure that this change does not affect the distribution of the h_ε and r_ε, up to a negligible error.

(2) The second change we make is that the garbled circuit $\tilde{P}^0$ is generated in simulated form instead of honestly. Note that at this point the distribution of this garbled circuit depends only on its output, which is $\{\mathsf{Enc}(\mathsf{k}_\varepsilon, (\mathsf{h}_\varepsilon, j, b), \mathsf{lab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}$, where $\{\mathsf{lab}_{j,b}\}_{j\in[\lambda], b\in\{0,1\}}$ are the input labels for the garbled circuit $\tilde{P}^1$.

(3) Observe that now the trapdoor t_ε is not being used at all and $\tilde{P}^0$ is simulated. Therefore, based on the security of the chameleon encryption, we have that for all $j \in [\lambda]$, $\mathsf{Enc}(\mathsf{k}_\varepsilon, (\mathsf{h}_\varepsilon, j, 1 - \mathsf{h}_{\mathsf{ID}[1],j}), \mathsf{lab}_{j,1-\mathsf{h}_{\mathsf{ID}[1],j}})$ hides $\mathsf{lab}_{j,1-\mathsf{h}_{\mathsf{ID}[1],j}}$. Hence, we can change the hardcoded ciphertexts from

$$\{\mathsf{Enc}(\mathsf{k}_\varepsilon, (\mathsf{h}_\varepsilon, j, b), \mathsf{lab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}$$

to

$$\{\mathsf{Enc}(\mathsf{k}_\varepsilon, (\mathsf{h}_\varepsilon, j, b), \mathsf{lab}_{j,\mathsf{h}_{\mathsf{ID}[1],j}})\}_{j\in[\lambda], b\in\{0,1\}}.$$

After this modification the encryptions only contain the λ labels encoding to $\mathsf{h}_{\mathsf{ID}[1]}$ rather than the full set of $2 \cdot \lambda$ labels.

(4) Finally, the fourth change we make is that we reverse the first change. In particular, we generate h_ε as is done in the real execution.

As a consequence, at this point only the labels $\{\mathsf{lab}_{j,\mathsf{h}_{\mathsf{ID}[1],j}}\}_{j\in[\lambda]}$ are revealed in an information theoretic sense. Subsequently, we can therefore repeat the same sequence of hybrids for the next garbled circuit $\tilde{P}^1$ and so on. The full proof of security is provided in Section 6.2.

2.3 Hierarchical Identity-based Encryption

We will now discuss how these ideas can be modified to obtain hierarchical IBE (HIBE). We will generally follow the same tree-based approach as in our IBE construction. However, HIBE needs an additional mechanism to to enable the delegation of keys. Our main idea is to modify our IBE construction in a way such that the key at every internal node in the tree can act as a master secret key for the corresponding sub-tree, or equivalently take the same role as root leafs in our IBE construction. Towards this goal, we will need a pseudorandom function with additional properties to provide a mechanism that allows delegation keys of keys.

Arrangement of the Keys. In our IBE construction above each key, k_i corresponds to a level of the tree. Since our goal is key delegation, this causes a problem. We cannot release the corresponding trapdoor of such a key, otherwise all nodes on the same layer would be compromised. Thus, we need to refine our construction to tie chameleon encryption keys to single nodes, rather than entire levels.

In our construction, we will choose a global key k_G for the chameleon encryption scheme such that the associated hash function is sufficiently compressing. Compression will be achieved using this single global key k_G. We will next explain which information we associate with every node in the tree.

Each node v in the tree will have three main descriptors and additionally three auxiliary descriptors. The three main descriptors are a key k_v for a chameleon encryption scheme, a hash value h_v computed under k_v, and a public key pk_v for a public key encryption scheme. The three auxiliary descriptors are a hash value h'_v under key k_G as well as random coins r_v and r'_v. We will now ex-

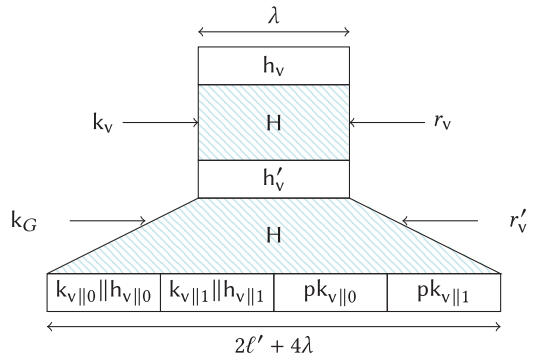

Fig. 2. The hashing relations for a node v in our HIBE construction. We separate the inputs to $\mathsf{H}(\mathsf{k}_G, \cdot)$ into four blocks. When we compute hash encryptions, they will address one of the four blocks.

plain how these values depend on one another. First, the hash-value h'_v is computed by hashing the main descriptors of the two child nodes $\mathsf{v}\|0$ and $\mathsf{v}\|1$ using the key k_G and random coins r'_v, i.e.

$$\mathsf{h}'_\mathsf{v} = \mathsf{H}(\mathsf{k}_G, \mathsf{k}_{\mathsf{v}\|0}\|\mathsf{h}_{\mathsf{v}\|0}\|\mathsf{k}_{\mathsf{v}\|1}\|\mathsf{h}_{\mathsf{v}\|1}\|\mathsf{pk}_{\mathsf{v}\|0}\|\mathsf{pk}_{\mathsf{v}\|1}; r'_\mathsf{v}).$$

The particular order in which we concatenate these values is chosen to simplify notation when we denote chameleon encryptions relative to this hash value. The hash value h_v in turn is obtained by hashing h'_v using the key k_v and random coins r_v, i.e.,

$$\mathsf{h}_\mathsf{v} = \mathsf{H}(\mathsf{k}_\mathsf{v}, \mathsf{h}'_\mathsf{v}; r_\mathsf{v}).$$

This arrangement is illustrated in Figure 2.

As in our IBE construction, we will define a local key lk_v, which in this case contains the main descriptors except pk_v and auxiliary descriptors of the node v as well as the main descriptors of the two child nodes $\mathsf{v}\|0$ and $\mathsf{v}\|1$, that is

$$\mathsf{lk}_\mathsf{v} = (\mathsf{k}_\mathsf{v}, \mathsf{h}_\mathsf{v}, r_\mathsf{v}, \mathsf{h}'_\mathsf{v}, r'_\mathsf{v}, \mathsf{k}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{k}_{\mathsf{v}\|1}, \mathsf{h}_{\mathsf{v}\|1}, \mathsf{pk}_{\mathsf{v}\|0}, \mathsf{pk}_{\mathsf{v}\|1}).$$

As in our IBE construction, we will generate node descriptions pseudorandomly. More specifically, a PRF will provide the random coins to generate the key k_v together with a collision-trapdoor t_v and the public key pk_v together with the a secret key sk_v. The hash value h_v will first be computed as a hash of 0^λ under the key k_v using random coins ω obtained from the PRF, i.e.,

$$\mathsf{h}_\mathsf{v} = \mathsf{H}(\mathsf{k}_\mathsf{v}, 0^\lambda; \omega).$$

Analogously as in our IBE construction, we can *connect* h_v and h'_v by using the Chameleon property. Specifically, we use the trapdoor t_v to compute random coins ω such that $\mathsf{h}_\mathsf{v} = H(\mathsf{k}_\mathsf{v}, \mathsf{h}'_\mathsf{v}; \omega)$.

A *simple* identity secret key sk'_ID for a node ID, i.e., an identity secret key that does not support delegation, consists of the local keys lk_v for all nodes v along the path from the root to ID as well as the decryption key dk_ID corresponding to the public key pk_ID at the node ID, i.e.,

$$\mathsf{sk}'_\mathsf{ID} = (\mathsf{lk}_\epsilon, \mathsf{lk}_{\mathsf{ID}[1]}, \ldots, \mathsf{lk}_{\mathsf{ID}[1..n]}, \mathsf{dk}_\mathsf{ID}).$$

Pseudorandom Functions with Delegation. We will now discuss the additional property we require of the pseudorandom functions in our construction. Let F be a function with the following property: On input a seed s and a binary string ID, it outputs a new seed s_ID and a string y, i.e., $(s_\mathsf{ID}, y) = F(s, \mathsf{ID})$. We will use the notation $s_\mathsf{ID} = F_1(s, \mathsf{ID})$ and $y = F_2(s, \mathsf{ID})$. We require that it holds for any ID and any ID′ of length at least 1 that $F(s_\mathsf{ID}, \mathsf{ID}') = F(s, \mathsf{ID}\|\mathsf{ID}')$. In other words, the *delegated seed* s_ID can be used to evaluate F on all inputs for which ID is a true prefix, using the same algorithm F.

In terms of security, we require the following selective security property. For every $\mathsf{ID}^* \in \{0,1\}^*$ of polynomial length n it holds that $(F_2(s, \mathsf{ID}[1..i]))_{i\in[n]}$ is pseudorandom, even given $F(s, \mathsf{ID}')$ for any ID' that is neither a prefix of ID nor ID itself.

Such a PRF can be constructed via the GGM paradigm, using essentially the same ideas as in a construction of *puncturable pseudorandom functions* [Boyle et al. 2014b]. Let $G : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ be a length-tripling pseudorandom generator. Write $G(s) = (G_0(s), G_1(s), G_2(s))$ where each $G_i(s)$ is of length λ. For a given ID, we compute $F(s, \mathsf{ID})$ by first computing $s_{\mathsf{ID}} := \mathsf{G}_{\mathsf{ID}[n]}(\mathsf{G}_{\mathsf{ID}[n-1]}(\ldots(\mathsf{G}_{\mathsf{ID}[1]}(s))\ldots))$, then $y := \mathsf{G}_2(s_{\mathsf{ID}})$ and setting the output to be (s_{ID}, y). Security of this construction under the selective security notion defined above follows via a simple hybrid argument using the security of the pseudorandom generator G.

When computing local keys lk_v as specified above, we will use $F_2(s,\cdot)$ as the pseudorandom function.

Master Public Keys, Master Secret Keys, and Identity Secret Keys. We can now specify how the master public keys, master secret keys, and identity secret keys for our HIBE scheme are computed. The master public key mpk consists of the global hash key k_G, the hashing key k_ϵ, and hash value h_ϵ at the root. The master secret key consists only of the PRF seed s used to compute the local keys. Finally, identity secret keys $\mathsf{sk}_{\mathsf{ID}}$ are computed as the simple identity secret keys specified above, but additionally include the value $s_{\mathsf{ID}} = F_1(s, \mathsf{ID})$, i.e.,

$$\mathsf{sk}_{\mathsf{ID}} = (\mathsf{lk}_\epsilon, \mathsf{lk}_{\mathsf{ID}[1]}, \ldots, \mathsf{lk}_{\mathsf{ID}[1..n]}, \mathsf{dk}_{\mathsf{ID}}, F_1(s, \mathsf{ID})).$$

Encryption and Decryption. For our HIBE scheme, encryption and decryption will be performed similar as in the IBE scheme above. The first difference is that as every node v is associated with a public key pk_v, we will be able to encrypt messages to every internal node v. Thus, we will refer to paths to such nodes as root-to-node paths instead of root-to-leaf paths. As a matter of fact, in the way we specified our scheme there are no leaf nodes, but the tree is essentially infinitely large.

The main difference in the encryption procedure is that the encryptor provides two garbled circuits $\tilde{P}^i$ and $\tilde{Q}^i$ for each node along the root-to-node path for an identity ID. As in the IBE scheme above, the decryptor will traverse the path by interleaving garbled circuit evaluations with steps in which it decrypts the chameleon encryption ciphertexts generated by the garbled circuits. There will also be a *terminator* circuit T that outputs an encryption of the message m under the node public key $\mathsf{pk}_{\mathsf{ID}}$.

For an identity ID of length n, then encryptor will generate $2n+1$ garbled circuits $\tilde{\mathsf{P}}^0, \tilde{\mathsf{Q}}^0, \ldots, \tilde{\mathsf{P}}^{n-1}, \tilde{\mathsf{Q}}^{n-1}$, and $\tilde{\mathsf{T}}$ such that a decryptor in possession of the identity secret key $\mathsf{sk}_{\mathsf{ID}}$ will be able to decrypt the message m. The encryptor will generate the garbled circuits in reverse order, i.e., it will first garble T, then Q^{n-1}, and so forth. We will now discuss the circuits P^i and Q^i in some more detail.

- The circuit P^i takes as input a node key k_v and a hash value h_v and generates chameleon encryptions of the input labels of the garbled circuit $\tilde{\mathsf{Q}}^i$ using k_v and h_v. To do so, P^i has the input labels for $\tilde{\mathsf{Q}}^i$ hardwired.
- The circuit Q^i takes as input a hash value h'_v and generates chameleon encryptions of the input labels of the next garbled circuit in the sequence, which is either $\tilde{\mathsf{P}}^{i+1}$ or $\tilde{\mathsf{T}}$. Consequently, the circuit Q^i has the input labels of the next garbled circuit hardwired. Moreover, it also has the global key k_G and the ith identity bit ID_i hardwired.
  Recall that the hash value h'_v is computed by $\mathsf{h}'_\mathsf{v} = \mathsf{H}(\mathsf{k}_G, (\mathsf{k}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{k}_{\mathsf{v}\|1}, \mathsf{h}_{\mathsf{v}\|1}, \mathsf{pk}_{\mathsf{v}\|0}, \mathsf{pk}_{\mathsf{v}\|1}); r')$. If the next circuit after Q^i is a circuit P^{i+1}, then the chameleon encryption generated by Q^i is such that the decryptor will be able to recover labels allowing him to eval-

uate $\tilde{\mathsf{P}}^{i+1}$ on input $\mathsf{k}_{\mathsf{v}\|\mathsf{ID}[i]}, \mathsf{h}_{\mathsf{v}\|\mathsf{ID}[i]}$. However, if the next circuit is T, then the chameleon encryption ciphertext is such that the decryptor will be able to evaluate $\tilde{\mathsf{T}}$ on input $\mathsf{pk}_{\mathsf{v}\|\mathsf{ID}[i]} = \mathsf{pk}_{\mathsf{ID}}$.

The final ciphertext consists of the input labels for $\tilde{\mathsf{P}}^0$ encoding the input $(\mathsf{k}_\epsilon, \mathsf{h}_\epsilon)$ and the sequence of garbled circuits $\tilde{\mathsf{P}}^0, \tilde{\mathsf{Q}}^0, \ldots, \tilde{\mathsf{P}}^{n-1}, \tilde{\mathsf{Q}}^{n-1}$, and $\tilde{\mathsf{T}}$. Now, a decryptor in possession of the identity secret key $\mathsf{sk}_{\mathsf{ID}}$ will be able to successively evaluate the garbled circuits provided in the ciphertext in very much the same fashion as in our IBE construction. The last garbled circuit $\tilde{\mathsf{T}}$ in the sequence will output a ciphertext c encrypting the message m under the node public key pk_{v}. The decryptor can now decrypt this ciphertext and recover m using the secret key $\mathsf{dk}_{\mathsf{ID}}$, which is contained in the secret key $\mathsf{sk}_{\mathsf{ID}}$.

Key Delegation. We will now discuss how we achieve the determining feature of HIBE, namely, the delegation of hierarchical keys. That is, we will show how to compute a key $\mathsf{sk}_{\mathsf{ID}\|\mathsf{ID}'}$ given a key $\mathsf{sk}_{\mathsf{ID}}$. Recall that an identity secret key $\mathsf{sk}_{\mathsf{ID}}$ consists of the *local keys* lk_v for all nodes v along the root-to-node path for ID, the secret key $\mathsf{dk}_{\mathsf{ID}}$ and a node-seed $s_{\mathsf{ID}} = F_1(s, \mathsf{ID})$.

The basic idea now is that we can treat the node ID essentially as the root-node of a subtree. The delegation property of the PRF lets us compute $F(s, \mathsf{ID}\|\mathsf{ID}')$ by $F(s_{\mathsf{ID}}, \mathsf{ID}')$. This allows us to compute $F_2(s, \mathsf{v})$ for any v that is prefixed by ID. Consequently, given s_{ID} one can compute all the local keys lk_{v} whenever ID is a prefix of v. Assume that ID' is of length m. We can compute delegated keys $\mathsf{sk}_{\mathsf{ID}\|\mathsf{ID}'}$ as follows: For $i = 1, \ldots, m$ compute $\mathsf{lk}_{\mathsf{ID}\|\mathsf{ID}'[1..i]}$. Furthermore, compute the secret key $\mathsf{dk}_{\mathsf{ID}\|\mathsf{ID}'}$. We can then set $\mathsf{sk}_{\mathsf{ID}}$ to

$$\mathsf{sk}_{\mathsf{ID}} = (\mathsf{lk}_{\mathsf{ID}[1]}, \ldots, \mathsf{lk}_{\mathsf{ID}[1..n]}, \mathsf{lk}_{\mathsf{ID}\|\mathsf{ID}'[1]}, \mathsf{lk}_{\mathsf{ID}\|\mathsf{ID}'[1..m]}, F(s_{\mathsf{ID}}, \mathsf{ID}'), \mathsf{dk}_{\mathsf{ID}\|\mathsf{ID}'}).$$

Note that the local keys $\mathsf{lk}_{\mathsf{ID}[1]}, \ldots, \mathsf{lk}_{\mathsf{ID}[1..n]}$ are simply taken from $\mathsf{sk}_{\mathsf{ID}}$. By the delegation property of the PRF, we would obtain exactly the same key if we generated it starting from the root node, correctness of the scheme follows.

Ideas of the Security Proof. For this scheme, we only prove selective security, i.e., the adversary announces the challenge identity ID^* before seeing any keys. This allows us to use the selective security property of the delegatable PRF and replace all values $(F_2(s, \mathsf{ID}^*[1..i]))_{i\in[n]}$ on the root-to-node path for the challenge identity ID^* by truly random values. In the remaining steps of the proof, we use security of the chameleon encryption scheme and of garbled circuit analogously as in the proof-sketch above. Specifically, in a sequence of hybrids, we use the security properties of garbled circuits and chameleon encryption in an alternating fashion. That is, we replace the garbled circuits in the challenge ciphertext $\tilde{\mathsf{P}}^0, \tilde{\mathsf{Q}}^0, \ldots, \tilde{\mathsf{P}}^{n-1}, \tilde{\mathsf{Q}}^{n-1}, \tilde{\mathsf{T}}$ by simulated garbled circuits, starting with $\tilde{\mathsf{P}}^0$. In the final hybrid all garbled circuits are simulated, and we can invoke the semantic security of the public key encryption scheme to argue that the challenge ciphertext is hidden.

We remark that this proof-strategy does not extend to fully secure HIBE from standard polynomial time assumptions. The main roadblock is essentially that we do not have constructions of adaptively secure delegatable pseudorandom functions.

3 PRELIMINARIES

Let λ denote the security parameter. We use the notation $[n]$ to denote the set $\{1, \ldots, n\}$. By PPT, we mean a probabilistic polynomial time algorithm. For any set S, we use $x \xleftarrow{\$} S$ to mean that x is sampled uniformly at random from the set S.[10] Alternatively, for any distribution D, we use $x \xleftarrow{\$} D$

[10] We use this notion only when the sampling can be done by a PPT algorithm and the sampling algorithm is implicit.

to mean that x is sampled from the distribution D. For a PPT algorithm A that takes input x, we will write $A(x;r)$ to denote that A uses explicit random coins r. We use the operator := to represent assignment and = to denote an equality check. We say that a function $\nu : \mathbb{N} \to \mathbb{R}_{>0}$ is *negligible*, if for every constant $c > 0$ it holds that $\nu(n) < 1/n^c$ for all but finitely many $n \in \mathbb{N}$. If a function $\epsilon : \mathbb{N} \to \mathbb{R}_{>0}$ is not negligbile, i.e., there exists a constant $c > 0$ such that $\epsilon(n) > 1/n^c$ for infinitely many $n \in \mathbb{N}$, then we say it is *non-negligible*.

3.1 Hard Problems

In this section, we will briefly review the computational assumptions relevant to this work. We start with the notion of a *hard-to-compute function*.

Definition 3.1 (Hard-to-Compute Functions). Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function and $X = \{X_\lambda\}_\lambda$ be an ensemble of input distributions for f. We say that f is hard-to-compute for X if it holds for every PPT algorithm $\mathcal{A}$ that

$$\Pr[\mathcal{A}(1^\lambda, x) = f(x)] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random choice of $x \xleftarrow{\$} X_\lambda$ and the random coins of $\mathcal{A}$.

The specific hard-to-compute function relevant to this work is the Diffie-Hellman function. We assume familiarity with basic group theory. We will first define the notion of a group generation algorithm, which is a possibly randomized algorithm that generates the description of a group. This notion is particularly useful when we consider groups that relate to the factoring assumption and the group in question cannot be fixed.

Definition 3.2 (Group Ensembles). A group ensemble is given by a PPT algorithm $\mathcal{G}$ with the following syntax. Upon input, a security parameter 1^λ $\mathcal{G}$ outputs the description of a cyclic group $\mathbb{G}$, its order $|\mathbb{G}| \geq \lambda$, and a generator $g \in \mathbb{G}$.

We can now define the Computational Diffie-Hellman assumption relative to a group generation algorithm $\mathcal{G}$.

Definition 3.3 (The Computational Diffie-Hellman (CDH) Problem). Let $\mathcal{G}$ be the sampler of a group ensemble. Let $(\mathbb{G}, q, g) := \mathcal{G}(1^\lambda)$ and $a, b \xleftarrow{\$} \mathbb{Z}_q$ be sampled uniformly at random from $\mathbb{Z}_q$. The computational Diffie-Hellman problem relative to $\mathcal{G}$, $\mathsf{CDH}(\mathcal{G})$, asks to compute the function $\mathsf{DH}_{\mathbb{G}}(g, g^a, g^b) = g^{ab}$. We say the $\mathsf{CDH}(\mathcal{G})$ problem is hard if the function DH is hard-to-compute in the sense of Definition 3.1 for the input distribution (g, g^a, g^b) for uniformly random $a, b \xleftarrow{\$} \mathbb{Z}_p$. Specifically, we say the problem $\mathsf{CDH}(\mathcal{G})$ is hard if it holds for every PPT algorithm $\mathcal{A}$ that

$$\Pr[\mathcal{A}(\mathbb{G}, g, g^a, g^b) = \mathsf{DH}(g, g^a, g^b)] \leq \mathsf{negl}(\lambda),$$

where $(\mathbb{G}, q, g) := \mathcal{G}(1^\lambda)$ and $a, b \xleftarrow{\$} \mathbb{Z}_q$.

Since all currently known attacks against the CDH problem proceed by solving the discrete logarithm problem, we provide this problem here for completeness.

Definition 3.4 (The Discrete Logarithm (DLOG) Problem). Let $\mathbb{G}$ be the sampler of a group ensemble. Let $(\mathbb{G}, q, g) := \mathcal{G}(1^\lambda)$ and $x \xleftarrow{\$} \mathbb{Z}_q$ be sampled uniformly at random from $\mathbb{Z}_q$. The discrete logarithm problem relative to $\mathcal{G}$, $\mathsf{DLOG}(\mathcal{G})$, asks to compute x given g^x. We say the $\mathsf{DLOG}(\mathcal{G})$ problem is hard if it holds for every PPT algorithm $\mathcal{A}$ that

$$\Pr[\mathcal{A}(\mathbb{G}, g, g^x) = x] \leq \mathsf{negl}(\lambda),$$

where $(\mathbb{G}, q, g) := \mathcal{G}(1^\lambda)$ and $x \xleftarrow{\$} \mathbb{Z}_q$.

The computational Diffie-Hellman problem $\mathsf{CDH}(\mathcal{G})$ directly reduces to the discrete logarithm problem $\mathsf{DLOG}(\mathcal{G})$: Given an algorithm $\mathcal{A}$ that solves the $\mathsf{DLOG}(\mathcal{G})$ problem, we can solve the $\mathsf{CDH}(\mathcal{G})$ problem by computing $a := \mathcal{A}(\mathbb{G}, g, g^a)$, $b := \mathcal{A}(\mathbb{G}, g, g^b)$ and outputting $g^{a \cdot b}$.

Notation. In the main body, we will assume that the group generation algorithm $\mathcal{G}$ will be run by a Setup or Gen algorithm to obtain the description of a group $\mathbb{G}$ and a generator g without explicitly stating so.

3.2 Candidate Groups

We will provide a non-exhaustive discussion of candidate group ensembles $\mathcal{G}$ for which the problem $\mathsf{CDH}(\mathcal{G})$ is conjectured to be hard.

- Prime Fields [Diffie and Hellman 1976]: In this instantiation, groups are sampled as follows: Sample a random prime q of bit-length $\Omega(\lambda)$ such that $q-1$ has a prime factor p of bit-length $\geq \Omega(\lambda)$. Then the multiplicative group $\mathbb{F}_q^*$ of the finite field $\mathbb{F}_q$ contains a cyclic subgroup $\mathbb{G}$ of order p. A generator g of $\mathbb{G}$ is sampled by choosing a random $x \in \mathbb{F}_q^*$ and setting $g := x^{(q-1)/p}$. Any such $g \neq 1$ is a generator of $\mathbb{G}$. Diffie and Hellman [1976] conjecture that the computational Diffie-Hellman problem $\mathsf{CDH}(\mathbb{G})$ is hard for such groups $\mathbb{G}$.
  As mentioned above, all known attacks on the CDH problem attack it via the discrete logarithm problem. The index calculus method [Adleman 1979; Hellman and Reyneri 1983; Pohlig 1977] has sub-exponential runtime $e^{O((\log q)^{\frac{1}{2}} \cdot (\log\log q)^{\frac{1}{2}})}$. The currently fastest discrete logarithm algorithms for arbitrary field size q, which includes the case when q is prime, are based on the the number-field sieve [Commeine and Semaev 2006; Gordon 1993; Joux and Lercier 2003; Joux et al. 2006], which in turn is based on the index-calculus method. These algorithms have subexponential runtime $e^{O((\log q)^{\frac{1}{3}} \cdot (\log\log q)^{\frac{2}{3}})}$. For the special case that the field size q is of the form p^n for a small prime p, a recent breakthrough [Barbulescu et al. 2014] obtained an algorithm with quasi-polynomial complexity $e^{O((\log\log(q))^2)}$. Consequently, groups in fields $\mathbb{F}_{p^n}$ are now considered outdated from a cryptanalytic perspective.
- The group of quadratic residues [Biham et al. 1997; Hofheinz and Kiltz 2009; McCurley 1988; Shmuely 1985]: For this instantiation, the group generation algorithm $\mathcal{G}$ chooses a random Blum integer $N = P \cdot Q$. That is, $P = 2p+1$ and $Q = 2q+1$ are random safe-primes (i.e., p, q are prime) of bit-length $\Omega(\lambda)$. It sets $\mathbb{G}$ to be the group of quadratic residues $\mathbb{QR}_N \subseteq \mathbb{Z}_N^*$, i.e.,
$$\mathbb{QR}_N = \{x^2 \in \mathbb{Z}_N^* \mid x \in \mathbb{Z}_N^*\}.$$
  The group $\mathbb{G} = \mathbb{QR}_N$ has order $|\mathbb{QR}_N| = \Phi(N)/4 = (P-1)(Q-1)/4 = p \cdot q$. Choosing a random g from this group yields a group generator, except with negligible probability $1/p + 1/q$. We note that in this case the group order $|\mathbb{QR}_N|$ is not made public. Nevertheless, we can sample statistically close to uniformly random in $\mathbb{G}$ by choosing a random $r \xleftarrow{\$} \mathbb{Z}_N$ and computing g^r.
  Shmuely [1985] and McCurley [1988] proved that for this choice the $\mathsf{CDH}(\mathcal{G})$ problem is at least as hard as the factorization problem FACT, given that the group order $|\mathbb{QR}_N|$ is *not* provided to the adversary (also see Biham et al. [1997] and Hofheinz and Kiltz [2009]).

  Definition 3.5 (The Factoring Problem). Given a random Blum integer $N = P \cdot Q$, the FACT problem asks to compute P and Q.

  Lemma 3.6. *Let the group generation algorithm $\mathcal{G}$ be as above. If there exists a PPT algorithm that solves the* $\mathsf{CDH}(\mathcal{G})$ *problem with non-negligible probability, then there exists an algorithm that solves the* FACT *problem with non-negligible probability.*

The converse is not known. On the contrary, even if the factorization $N = (2p + 1) \cdot (2q + 1)$ is known, it can be shown via the Chinese Remaindering Theorem that solving the CDH problem in $\mathbb{QR}_N$ is still equivalent to solving CDH in two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ simultaneously, where $\mathbb{G}_1 \subseteq \mathbb{F}_P^*$ is of order p and $\mathbb{G}_1 \subseteq \mathbb{F}_Q^*$ is of order q.
Consequently, the best-known algorithms for this instantiation are essentially the same as in the case of prime-order subgroups in fields of prime size.

- Elliptic Curve Groups [Koblitz 1987; Miller 1986]. Let $\mathbb{Z}_q$ be a finite field of characteristic p. An elliptic curve $E(\mathbb{Z}_q)$ consists of all points $(x, y) \in \mathbb{Z}_q^2$ satisfying a Weierstrass relation

$$y^3 = x^3 + Ax + B$$

and a special *point at infinity* $\mathcal{O}$. We assume that the polynomial $x^3 + Ax + B$ has non-zero discriminant (i.e., $4A^3 + 27B^2 \neq 0$). We can define a group-law + on $E(\mathbb{Z}_q)$ by setting $P_1 + P_2$ to be the unique point $P_3 = (x, y) \in E(\mathbb{Z}_q)$, such that P_1, P_2 and $(x, -y)$ are co-linear (i.e., they are on a straight line). For the special point $\mathcal{O}$, we define $P + \mathcal{O} = P$, i.e., $\mathcal{O}$ is the identity element. For a prime q, the Hasse bound [Hasse 1936; Niederreiter and Xing 2009] provides an estimate for $|E(\mathbb{Z}_p)|$ by $||E(\mathbb{Z}_p)| - p - 1| \leq 2\sqrt{p}$, i.e., the order of the group is roughly $p + 1$. Such curves can, e.g., be sampled by choosing $A, B \in \mathbb{Z}_p$ uniformly random subject to the constraint $4A^3 + 27B^2 \neq 0$. While the group operation in such groups is typically written additively, changing to the multiplicative notation we use in this work is just a syntactic replacement.

When p is a large prime, the best-known attacks against the discrete logarithm problem in this group are generic and have exponential overhead. Specifically, for a group $\mathbb{G}$ of order $\approx 2^\lambda$, the baby-step giant-step algorithm [Cohen 2013; Shanks 1972] has both time and space complexity $O(2^{\lambda/2})$. Pollards ρ method [Pollard 1978] improves this to time complexity $O(2^{\lambda/2})$ and space complexity $\mathsf{poly}(\lambda)$.

However, if an elliptic curve group $\mathbb{G}$ is equipped with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ where the group $\mathbb{G}_T$ is a subgroup of the multiplicative group $\mathbb{F}_p^*$ of a finite field $\mathbb{F}_p$ [Joux 2000], then the DLOG problem in $\mathbb{G}$ immediately reduces to the DLOG problem in $\mathbb{G}_T$. Since discrete logarithms in such groups $\mathbb{G}_T$ can be computed in sub-exponential time (see the discussion above), this implies that discrete logarithms in $\mathbb{G}$ can also be computed in sub-exponential time. Consequently, by the current state of knowledge, elliptic curve groups that are equipped with pairings must be regarded as weaker than curves without a pairing.

3.3 Hardcore Predicates

Security notions for public-key encryption (cf. Section 3.7) are indistinguishability-based. To base the security of a public-key encryption scheme on a search assumption, we need to first transform a search problem into a decisional problem. Hardcore predicates are a cryptographic standard tool that provide such a transformation.

Definition 3.7 (Hardcore Predicates). Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function and $X = \{X_\lambda\}$ be an ensemble of input distributions for f. We say an efficiently computable function $\mathsf{HardCore} : \{0,1\}^* \to \{0,1\}$ is a hardcore predicate for f and X, if it holds for every PPT algorithm $\mathcal{A}$ that

$$\Pr[\mathcal{A}(1^\lambda, x) = \mathsf{HardCore}(f(x))] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $x \xleftarrow{\$} X_\lambda$. The probability is taken over the random choice of x and the random coins of $\mathcal{A}$.

For any hard-to-compute function f, the Goldreich-Levin hardcore predicate yields another hard-to-compute function g, which is only a gentle modification of f, such that g has an explicit hardcore predicate. Currently, the Goldreich-Levin hardcore predicate (or variants thereof) provides the only means to construct public-key encryption from the computational Diffie-Hellman assumption in the standard model.[11]

Theorem 3.8 (Goldreich-Levin Hardcore Predicate [Akavia et al. 2003; Goldreich and Levin 1989; Kushilevitz and Mansour 1991]). *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a hard-to-compute function for an ensemble $X = \{X_\lambda\}_\lambda$. There exists an explicit efficiently computable predicate $\mathsf{GL} : \{0,1\}^* \to \{0,1\}$ such that GL is hardcore for the function g where $g(x,r) = (f(x), r)$ with the ensemble $X' = \{X_\lambda \times U_{\ell(\lambda)}\}$.*

3.4 Pseudorandom Generators and Pseudorandom Functions

Pseudorandom Generators. Pseudorandom generators are a standard primitive that allow expanding a short random string s into a long *random looking* string $G(s)$.

Definition 3.9 (Pseudorandom Generators). Let ℓ be a polynomial and let $G : \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ be an efficiently computable function that maps a seed $s \in \{0,1\}^\lambda$ to a string $G(s) \in \{0,1\}^{\ell(\lambda)}$. We say that G is a pseudorandom generator if it holds for any PPT distinguisher $\mathcal{D}$ that

$$\left|\Pr[\mathcal{D}(1^\lambda, G(s)) = 1] - \Pr[\mathcal{D}(1^\lambda, r) = 1]\right| < \mathsf{negl}(\lambda),$$

where $r \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$ is chosen uniformly random. The first probability is taken over the random choice of the seed s and the random coins of the distinguisher $\mathcal{D}$, whereas the second probability is taken over the random choice of r and the random coins of $\mathcal{D}$.

Pseudorandom generators can be constructed from any one-way function [Håstad et al. 1999; Impagliazzo et al. 1989].

Pseudorandom Functions. Pseudorandom functions are efficiently computable (seeded) functions, which are indistinguishable from truly random functions under oracle access.

Definition 3.10 (Pseudorandom Functions). An efficiently computable function $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ is called *pseudorandom function* if it holds for any PPT distinguisher $\mathcal{D}$ with oracle access to a function $\{0,1\}^* \to \{0,1\}^*$ that

$$\left|\Pr[\mathcal{D}^{\mathsf{PRF}(s,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{D}^{f(\cdot)}(1^\lambda) = 1]\right| < \mathsf{negl}(\lambda),$$

where $f : \{0,1\}^* \to \{0,1\}^*$ is a uniformly random function. The first probability is taken over the choice of the seed s and the random coins of $\mathcal{D}$, whereas the second probability is taken over the random choice of f and the random coins of $\mathcal{D}$.

Pseudorandom functions can be constructed from any pseudorandom generator [Goldreich et al. 1984] and consequently from any one-way function.

Delegatable Pseudorandom Functions. In our HIBE construction, we will need a pseudorandom function for which the inputs can be binary strings of unrestricted length and which supports the delegation of seeds for inputs that start certain prefixes.

Definition 3.11. A delegatable pseudorandom function consists of two algorithms PRF and $\mathsf{PRF.Delegate}$ with the following syntax:

[11] That is, without, e.g., resorting to the random oracle heuristic [Bellare and Rogaway 1993].

- $\mathsf{PRF}_s(x)$ takes as input a seed $s \in \{0,1\}^\lambda$ as and a string $x \in \{0,1\}^*$ and outputs a value $u \in \{0,1\}^\lambda$.
- $\mathsf{PRF.Delegate}(s, x)$ takes as input a seed s and an input x and outputs a seed s_x.

We require the following properties of a delegatable pseudorandom function:

- **Delegatability:** It holds for all inputs $x, x' \in \{0,1\}^*$ that
$$\mathsf{PRF}(s, x\|x') = \mathsf{PRF}(s_x, x'),$$
where $s_x := \mathsf{PRF.Delegate}(s, x)$.
- **Pseudorandomness:** It holds for all PPT distinguishers $\mathcal{D}$ and every $x \in \{0,1\}^*$ of size at most polynomial in λ that
$$|\Pr[\mathcal{D}^{\mathsf{PRF}(s,\cdot),\mathsf{Delegate}(s,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{D}^{H(\cdot),\mathsf{Delegate}(s,\cdot)}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda),$$
where $s \xleftarrow{\$} \{0,1\}^\lambda$ is chosen uniformly at random, H is a function that is uniformly random on all prefixes of x (including x) and identical to $\mathsf{PRF}(s,\cdot)$ on all other inputs, and $\mathsf{Delegate}(s,\cdot)$ delegates seeds for all inputs $x' \in \{0,1\}^*$ that are not a prefix of x.

We will briefly sketch a simple variant of the GGM construction [Goldreich et al. 1984] that satisfies the above definition. Let $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ be a length-tripling pseudorandom generator and G_0, G_1, and G_2 be the $1 \ldots \lambda$, $\lambda + 1 \ldots 2\lambda$ and $2\lambda + 1 \ldots 3\lambda$ bits of the output of G, respectively. Now define a GGM-type pseudo-random function $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ such that $\mathsf{PRF}(s, x) := \mathsf{G}_2(\mathsf{G}_{x_n}(\mathsf{G}_{x_{n-1}}(\ldots(\mathsf{G}_{x_1}(s))\ldots)))$, where for each $i \in [n]$ x_i is the ith bit of $x \in \{0,1\}^n$. $\mathsf{PRF.Delegate}(s, x)$ computes and outputs $\mathsf{G}_{x_n}(\mathsf{G}_{x_{n-1}}(\ldots(\mathsf{G}_{x_1}(s))\ldots))$. Security of this construction follows via a simple hybrid argument.

3.5 Garbled Circuits

Garbled circuits were first introduced by Yao [1982] (see also Lindell and Pinkas [2009] and Bellare et al. [2012] for a detailed proof and further discussion). A circuit garbling scheme is a tuple of PPT algorithms (GCircuit, Eval). Conceptually, garbled circuit reduces the problem of securely computing any circuit to the problem of securely computing a very simple projection function. Specifically, the GCircuit algorithm takes as input a circuit C with n input wires and outputs a *garbled circuit* $\widetilde{\mathsf{C}}$ together with $2n$ *keys* $(\mathsf{key}_{i,b})_{i\in[n],b\in\{0,1\}}$ (also called labels). An input $x \in \{0,1\}^n$ for the circuit C can now be *encoded* by computing $\tilde{x} = (\mathsf{key}_{i,x_i})_{i\in[n]}$. The correctness property requires that evaluating a garbled circuit on labels $\tilde{x} = (\mathsf{key}_{i,x_i})_{i\in[n]}$ using the Eval algorithm results in the same value as $\mathsf{C}(x)$. Security of garbling schemes is defined via a simulation-based notion. Specifically, we require the existence of a simulator Sim, which takes as input y and outputs a garbled circuit $\widetilde{\mathsf{C}}$ and keys $(\mathsf{key}_i)_{i\in[n]}$ such that for any input x
$$(\widetilde{\mathsf{C}}, (\mathsf{key}_{i,x_i})_i) \approx_C \mathsf{Sim}(\mathsf{C}(x)),$$
where $(\widetilde{\mathsf{C}}, (\mathsf{key}_{i,b})_{i,b}) := \mathsf{GCircuit}(\mathsf{C}(x))$. In other words, the garbled circuit $\widetilde{\mathsf{C}}$ together with the labels encoding x can be simulated given only $\mathsf{C}(x)$ but without any further knowledge of x. Consequently, to compute the circuit C securely, we only need to be able to securely compute the encoding function
$$\mathsf{Encode}(x, (\mathsf{key}_{i,b})_{i\in[n],b\in\{0,1\}}) = (\mathsf{key}_{i,x_i})_{i\in[n]}.$$
Garbled circuits were originally proposed to realize secure two-party computation [Yao 1986]. Say, two parties A and B want to securely evaluate a circuit $\mathsf{C}(x_a, x_b)$ on their secret inputs a and b, i.e., they want to compute $\mathsf{C}(a, b)$. This can be achieved as follows: Party B garbles the circuit C, obtaining a garbled circuit $\widetilde{\mathsf{C}}$ and labels for the inputs x_a and x_b. B encodes its input b using the

labels for x_b. To encode A's input a without B learning it, A and B can use an *oblivious transfer* protocol [Aiello et al. 2001; Bellare and Micali 1990; Peikert et al. 2008; Rabin 2005] that essentially allows to compute the above Encode function securely. B now sends the garbled circuit $\widetilde{\mathsf{C}}$ together with an encoding of his input to A, who can evaluate $\widetilde{\mathsf{C}}$ and learn $\mathsf{C}(a, b)$. Security against B is provided by the receiver security of the oblivious transfer protocol, whereas security against A is provided by the sender security of the oblivious transfer protocol and the security of the garbling scheme. We now provide the formal definition of garbled circuits using the formalism of Bellare et al. [2012]. A garbling scheme consists of a pair of PPT algorithms (GCircuit, Eval) such that the following holds:

- $(\widetilde{\mathsf{C}}, \{\mathsf{key}_{w,b}\}_{w\in\mathsf{inp}(\mathsf{C}),b\in\{0,1\}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{C})$: GCircuit takes as input a security parameter λ and a circuit C. This procedure outputs a *garbled circuit* $\widetilde{\mathsf{C}}$ and labels $\{\mathsf{key}_{w,b}\}_{w\in\mathsf{inp}(\mathsf{C}),b\in\{0,1\}}$ where each $\mathsf{key}_{w,b} \in \{0,1\}^\lambda$.[12]
- $y := \mathsf{Eval}(\widetilde{\mathsf{C}}, \{\mathsf{key}_{w,x_w}\}_{w\in\mathsf{inp}(\mathsf{C})})$: Given a garbled circuit $\widetilde{\mathsf{C}}$ and a garbled input represented as a sequence of input labels $\{\mathsf{key}_{w,x_w}\}_{w\in\mathsf{inp}(\mathsf{C})}$, Eval outputs an output y.

Correctness. For correctness, we require that for any circuit C and input $x \in \{0,1\}^m$ (here m is the input length to C), we have that:

$$\Pr\left[\mathsf{C}(x) = \mathsf{Eval}\left(\widetilde{\mathsf{C}}, \{\mathsf{key}_{w,x_w}\}_{w\in\mathsf{inp}(\mathsf{C})}\right)\right] = 1,$$

where $(\widetilde{\mathsf{C}}, \{\mathsf{key}_{w,b}\}_{w\in\mathsf{inp}(\mathsf{C}),b\in\{0,1\}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{C})$.

Security. For security, we require that there is a PPT simulator Sim such that for any C, x, we have that

$$\left(\widetilde{\mathsf{C}}, \{\mathsf{key}_{w,x_w}\}_{w\in\mathsf{inp}(\mathsf{C})}\right) \overset{\mathsf{comp}}{\approx} \mathsf{Sim}\left(1^\lambda, \mathsf{C}(x)\right),$$

where $(\widetilde{\mathsf{C}}, \{\mathsf{key}_{w,b}\}_{w\in\mathsf{inp}(\mathsf{C}),b\in\{0,1\}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{C})$.[13]

3.6 Public Key Encryption

As our constructions will make use of a public key encryption scheme, we provide the definitions here.

Definition 3.12 (Public Key Encryption). A public key encryption scheme consists of three PPT algorithms (G, E, D) with the following syntax:

- $\mathsf{G}(1^\lambda)$ takes as input a security parameter 1^λ and outputs a pair of public and secret keys $(\mathsf{pk}, \mathsf{dk})$.
- $\mathsf{E}(\mathsf{pk}, \mathsf{m})$ takes as input a public key pk and a plaintext m and outputs a ciphertext c.
- $\mathsf{D}(\mathsf{dk}, c)$ takes as input a secret key dk and a ciphertext c and outputs a plaintext m.

We require the following properties to hold:

- **Completeness:** For every security parameter λ and for all messages m, it holds that

$$\mathsf{D}(\mathsf{dk}, \mathsf{E}(\mathsf{pk}, \mathsf{m})) = \mathsf{m},$$

where $(\mathsf{pk}, \mathsf{dk}) := \mathsf{G}(1^\lambda)$.

[12] Typical definitions of garbled circuits do not require the length of each input label to be λ bits long. This additional requirement is crucial in our constructions as we chain garbled circuits. Note that input labels in any garbled circuit construction can always be shrunk to λ bits using a pseudorandom function.

[13] In abuse of notation, we assume that Sim knows the (non-private) circuit C. When C has (private) hardwired inputs, we assume that the labels corresponding to these are included in the garbled circuit $\tilde{C}$.

Experiment IND-CPA($\mathcal{A}$):

(1) $(\mathsf{pk}, \mathsf{dk}) \xleftarrow{\$} \mathsf{G}(1^\lambda)$
(2) $(\mathsf{m}_0, \mathsf{m}_1) \xleftarrow{\$} \mathcal{A}_1(\mathsf{pk})$.
(3) $b \xleftarrow{\$} \{0,1\}$.
(4) $\mathsf{m}^* := \mathsf{m}_b$
(5) $c^* \xleftarrow{\$} \mathsf{E}(\mathsf{pk}, \mathsf{m}^*)$
(6) $b' \xleftarrow{\$} \mathcal{A}_2(\mathsf{pk}, c^*)$
(7) Output 1 if $b = b'$ and 0 otherwise.

Fig. 3. Thesemantic security experiment.

- **Semantic Security (IND-CPA Security):** For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds:

$$\Pr[\text{IND-CPA}(\mathcal{A}) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where IND-CPA($\mathcal{A}$) is shown in Figure 3.

3.7 Identity-based Encryption

Identity-based encryption is a type of public key encryption for which only a single public key, the so-called master public key mpk, is necessary to send encrypted messages to all users across a system. The number of users can potentially be exponential, whereas the size master public key is required to be small and in particular independent of the number of users. Because of this property, the master public mpk can be seen as a *compressed representation* of an exponential number of user public keys. However, instead of generating their identity secret keys themselves, users obtain their identity secret keys $\mathsf{sk}_{\mathsf{ID}}$ from a key authority that holds a master secret key msk. To encrypt a message to a user with the identity ID, only the master public key mpk and the identity ID are needed. Such a ciphertext can then be decrypted using the identity secret key $\mathsf{sk}_{\mathsf{ID}}$. In terms of security, it is intuitively required that a ciphertext encrypted to an identity ID^* hides the encrypted message m, even for an adversary who holds an arbitrary number of identity secret keys $\mathsf{sk}_{\mathsf{ID}}$ for $\mathsf{ID} \neq \mathsf{ID}^*$.

A significant drawback when using IBE in its natural use-case for key management is that the key authority is a single point of failure. That is, if an adversary manages to compromise the key-authority, then security for all users in the system is lost. However, in recent years IBE has emerged as more of a tool that is used in the construction of sophisticated cryptographic primitives. Notable examples are constructions of chosen-ciphertext secure public key encryption [Canetti et al. 2004] and garbled RAM [Gentry et al. 2014]. In a way, these applications use IBE as an *access control mechanism.*

Below, we provide the formal definition of identity-based encryption (IBE).

Definition 3.13 (Identity-Based Encryption (IBE) [Boneh and Franklin 2001; Shamir 1984]). An *identity-based encryption* scheme consists of four PPT algorithms (Setup, KeyGen, Encrypt, Decrypt) defined as follows:

- $\mathsf{Setup}(1^\lambda)$: given the security parameter, it outputs a master public key mpk and a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{ID})$: given the master secret key msk and an identity $\mathsf{ID} \in \{0,1\}^n$, it outputs a decryption key $\mathsf{sk}_{\mathsf{ID}}$.

Experiment $\mathsf{IND}^{\mathsf{IBE}}_{\mathcal{A}}(1^\lambda)$:

(1) $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$.

(2) $(\mathsf{ID}^*, \mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{msk},.)}(\mathsf{mpk})$ where $|m_0| = |m_1|$ and for each query ID by $\mathcal{A}_1$ to $\mathsf{KeyGen}(\mathsf{msk},.)$ we have that $\mathsf{ID} \neq \mathsf{ID}^*$.

(3) $b \xleftarrow{\$} \{0,1\}$.

(4) $\mathsf{ct}^* \xleftarrow{\$} \mathsf{Encrypt}(\mathsf{mpk}, \mathsf{ID}^*, \mathsf{m}_b)$.

(5) $b' \xleftarrow{\$} \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},.)}(\mathsf{mpk}, \mathsf{ct}^*, \mathsf{st})$ where for each query ID by $\mathcal{A}_2$ to $\mathsf{KeyGen}(\mathsf{msk},.)$ we have that $\mathsf{ID} \neq \mathsf{ID}^*$.

(6) Output 1 if $b = b'$ and 0 otherwise.

Fig. 4. The $\mathsf{IND}^{\mathsf{IBE}}_{\mathcal{A}}$ experiment.

- $\mathsf{Encrypt}(\mathsf{mpk}, \mathsf{ID}, \mathsf{m})$: given the master public key mpk, an identity $\mathsf{ID} \in \{0,1\}^n$, and a message m, it outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{ID}}, \mathsf{ct})$: given a secret key $\mathsf{sk}_{\mathsf{ID}}$ for identity ID and a ciphertext ct, it outputs a string m.

The following completeness and security properties must be satisfied:

- **Completeness:** For all security parameters λ, identities $\mathsf{ID} \in \{0,1\}^n$ and messages m, the following holds:

$$\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{ID}}, \mathsf{Encrypt}(\mathsf{mpk}, \mathsf{ID}, \mathsf{m})) = \mathsf{m},$$

  where $\mathsf{sk}_{\mathsf{ID}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{ID})$ and $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- **Security:** For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(.)$ such that the following holds:

$$\Pr[\mathsf{IND}^{\mathsf{IBE}}_{\mathcal{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

  where $\mathsf{IND}^{\mathsf{IBE}}_{\mathcal{A}}$ is shown in Figure 4. We require that for each key query ID that $\mathcal{A}$ sends to the KeyGen oracle, it must hold that $\mathsf{ID} \neq \mathsf{ID}^*$.

Hierarchical Identity-Based Encryption (HIBE). A hierarchical IBE or HIBE scheme has a similar syntax as IBE. However, a HIBE scheme comes with the additional functionality that anyone who holds a secret key $\mathsf{sk}_{\mathsf{ID}}$ for an identity ID can compute identity secret keys $\mathsf{sk}_{\mathsf{ID}'}$ for identities that are *prefixed* by ID, i.e., ID is a prefix of ID′. This form of delegation adds significant functionality to IBE, as illustrated in the introduction. Roughly speaking, the security requirement for HIBE is that encryptions under an identity ID^* are secure as long as the adversary is not given identity secret keys for ID^* or any prefix of ID^*. In this work, we will only consider selective security for HIBE. This means that the adversary must choose the *challenge identity* ID^* before seing the master public key mpk or any identity secret keys. We remark that while several works achieve selectively secure HIBE in the standard model under various assumptions [Agrawal et al. 2010a, 2010b; Agrawal and Boyen 2009; Boneh and Boyen 2004a; Boneh et al. 2005; Cash et al. 2010; Gentry and Halevi 2009; Gentry and Silverberg 2002; Horwitz and Lynn 2002; Shi and Waters 2008], constructions of fully secure HIBE under standard assumptions are only known from the bilinear Diffie-Hellman assumption [Lewko and Waters 2010; Waters 2009]. Full security guarantees that encryptions under the challenge identity are secure even if the adversary has chosen the challenge identity adaptively, depending on the master secret key and identity key queries.

Experiment $\mathrm{IND}^{\mathrm{HIBE}}_{\mathcal{A}}(1^\lambda)$:

(1) $\mathsf{ID}^* \xleftarrow{\$} \mathcal{A}_1(1^\lambda)$
(2) $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$.
(3) $(\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},.)}$ where $|m_0| = |m_1|$ and for each query ID by $\mathcal{A}_2$ to KeyGen(msk, .) we require that ID is neither a prefix of ID^* nor ID^* itself.
(4) $b \xleftarrow{\$} \{0,1\}$.
(5) $\mathsf{ct}^* \xleftarrow{\$} \mathsf{Encrypt}(\mathsf{mpk}, \mathsf{ID}^*, \mathsf{m}_b)$.
(6) $b' \xleftarrow{\$} \mathcal{A}_3^{\mathsf{KeyGen}(\mathsf{msk},.)}(\mathsf{mpk}, \mathsf{ct}^*, \mathsf{st})$ where for each query ID by $\mathcal{A}_3$ to KeyGen(msk, .) we require that ID is neither a prefix of ID^* nor ID^* itself.
(7) Output 1 if $b = b'$ and 0 otherwise.

Fig. 5. The $\mathrm{IND}^{\mathrm{HIBE}}_{\mathcal{A}}$ Experiment.

Besides the aforementioned construction of forward secure public key encryption from HIBE [Canetti et al. 2003], HIBE gives rise to significantly more powerful access control mechanisms than standard IBE. For instance, consider a HIBE encrypted file system where each file is encrypted under its *path*, i.e., a file `C:\ documents\ file.txt` is encrypted under the identity $\mathsf{ID} =$ ‘‘`C:\ documents\ file.txt`’’. By giving a user a HIBE identity secret key corresponding to a directory, the user will be able to decrypt and read all files in this directory and its subdirectories by computing the corresponding delegated secret keys. We now provide the formal definition of HIBE.

A HIBE scheme is an IBE scheme except that we set $\mathsf{sk}_\varepsilon := \mathsf{msk}$ and modify the KeyGen algorithm. In particular, KeyGen takes $\mathsf{sk}_{\mathsf{ID}}$ and a string ID′ as input and outputs a secret key $\mathsf{sk}_{\mathsf{ID}\|\mathsf{ID}'}$. More formally:

- $\mathsf{KeyGen}(\mathsf{sk}_{\mathsf{ID}}, \mathsf{ID}')$: given the secret key $\mathsf{sk}_{\mathsf{ID}}$ and an identity $\mathsf{ID}' \in \{0,1\}^*$, it outputs a decryption key $\mathsf{sk}_{\mathsf{ID}\|\mathsf{ID}'}$.

The correctness condition for HIBE is same as for IBE, but we additionally require that identity keys computed using delegated keys are the same as the identity keys computed using the master secret key, i.e., for all ID and ID′ it holds that

$$\mathsf{KeyGen}(\mathsf{sk}_\varepsilon, \mathsf{ID}\|\mathsf{ID}') = \mathsf{KeyGen}(\mathsf{KeyGen}(\mathsf{sk}_\varepsilon, \mathsf{ID}), \mathsf{ID}').$$

Security is defined analogously to $\mathrm{IND}^{\mathrm{IBE}}_{\mathcal{A}}(1^\lambda)$, except that now we only consider the notion of *selective security* for HIBE—namely, the adversary $\mathcal{A}$ is required to announce the challenge identity ID^* before it has seen the mpk and has made any secret key queries. This experiment $\mathrm{IND}^{\mathrm{HIBE}}_{\mathcal{A}}$ is shown formally in Figure 5.

4 CHAMELEON ENCRYPTION

In this section, we provide the definition of a chameleon encryption scheme. Chameleon encryption will be a critical component in our IBE and HIBE constructions in Sections 6 and 7

Definition 4.1 (Chameleon Encryption). A chameleon encryption scheme consists of five PPT algorithms Gen, H, H^{-1}, Enc, and Dec with the following syntax:

- $\mathsf{Gen}(1^\lambda, n)$: Takes the security parameter λ and a message-length n (with $n = \mathsf{poly}(\lambda)$) as input and outputs a key k and a trapdoor t.

Experiment $\mathrm{IND}^{\mathrm{CE}}_{\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)}(1^\lambda)$:

(1) $(\mathsf{k}, \mathsf{t}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, n)$.
(2) $(\mathsf{x}, r, i \in [n], \mathsf{st}) \xleftarrow{\$} \mathcal{A}_1(\mathsf{k})$.
(3) $b \xleftarrow{\$} \{0,1\}$.
(4) $\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{k}, (\mathsf{H}(\mathsf{k}, \mathsf{x}; r), i, 1 - \mathsf{x}_i), b)$.
(5) $b' \xleftarrow{\$} \mathcal{A}_2(\mathsf{k}, \mathsf{ct}, (\mathsf{x}, r), \mathsf{st})$.
(6) Output 1 if $b = b'$ and 0 otherwise.

Fig. 6. The $\mathrm{IND}^{\mathrm{CE}}_{\mathcal{A}}$ experiment.

- $\mathsf{H}(\mathsf{k}, \mathsf{x}; r)$: Takes a key k, a message $\mathsf{x} \in \{0,1\}^n$, and coins r, and outputs a hash value h, where h is λ bits.
- $\mathsf{H}^{-1}(\mathsf{t}, (\mathsf{x}, r), \mathsf{x}')$: Takes a trapdoor t, previously used message $\mathsf{x} \in \{0,1\}^n$ and coins r, and a message $\mathsf{x}' \in \{0,1\}^n$ as input and returns r'.
- $\mathsf{Enc}(\mathsf{k}, (\mathsf{h}, i, b), \mathsf{m})$: Takes a key k, a hash value h, an index $i \in [n]$, $b \in \{0,1\}$, and a message $\mathsf{m} \in \{0,1\}^*$ as input and outputs a ciphertext ct.[14]
- $\mathsf{Dec}(\mathsf{k}, (\mathsf{x}, r), \mathsf{ct})$: Takes a key k, a message x, coins r, and a ciphertext ct as input and outputs a value m (or $\perp$).

We require the following properties[15]:

- **Uniformity:** For $\mathsf{x}, \mathsf{x}' \in \{0,1\}^n$, we have that the two distributions $\mathsf{H}(\mathsf{k}, \mathsf{x}; r)$ and $\mathsf{H}(\mathsf{k}, \mathsf{x}'; r')$ are statistically close (when r, r' are chosen uniformly at random).
- **Trapdoor Collisions:** For every choice of $\mathsf{x}, \mathsf{x}' \in \{0,1\}^n$ and r it holds that if $(\mathsf{k}, \mathsf{t}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, n)$ and $r' := \mathsf{H}^{-1}(\mathsf{t}, (\mathsf{x}, r), \mathsf{x}')$, then
$$\mathsf{H}(\mathsf{k}, \mathsf{x}; r) = \mathsf{H}(\mathsf{k}, \mathsf{x}'; r'),$$
i.e., $\mathsf{H}(\mathsf{k}, \mathsf{x}; r)$ and $\mathsf{H}(\mathsf{k}, \mathsf{x}'; r')$ generate the same hash h. Moreover, if r is chosen uniformly at random, then r' is also statistically close to uniform.
- **Correctness:** For any choice of $\mathsf{x} \in \{0,1\}^n$, coins r, index $i \in [n]$, and message m, it holds that if $(\mathsf{k}, \mathsf{t}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, n)$, $\mathsf{h} := \mathsf{H}(\mathsf{k}, \mathsf{x}; r)$, and $\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{k}, (\mathsf{h}, i, \mathsf{x}_i), \mathsf{m})$, then $\mathsf{Dec}(\mathsf{k}, (\mathsf{x}, r), \mathsf{ct}) = \mathsf{m}$.
- **Security:** For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds:
$$\Pr[\mathrm{IND}^{\mathrm{CE}}_{\mathcal{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$
where $\mathrm{IND}^{\mathrm{CE}}_{\mathcal{A}}$ is shown in Figure 6.

5 CONSTRUCTIONS OF CHAMELEON ENCRYPTION FROM CDH

Let $(\mathbb{G}, \cdot)$ be a cyclic group of order p (not necessarily prime) with generator g. Let $\mathsf{Sample}(\mathbb{G})$ be a PPT algorithm such that its output is statistically close to a uniform element in $\mathbb{Z}_p$, where p

[14] ct is assumed to contain (h, i, b).
[15] Typically, Chameleon Hash functions are defined to also have the collision resilience property. This property is implied by the semantic security requirement below. However, we do not need this property directly. Therefore, we do not explicitly define it here.

(not necessarily prime) is the order of $\mathbb{G}$.[16] We will now describe a chameleon encryption scheme assuming that the DH($\mathbb{G}$) problem is hard.

- Gen($1^\lambda, n$): For each $j \in [n]$, choose uniformly random values $\alpha_{j,0}, \alpha_{j,1} \xleftarrow{\$} \text{Sample}(\mathbb{G})$ and compute $g_{j,0} := g^{\alpha_{j,0}}$ and $g_{j,1} := g^{\alpha_{j,1}}$. Output (k, t) where[17]

$$\mathsf{k} := \left(g, \begin{pmatrix} g_{1,0}, g_{2,0} \ldots, g_{n,0} \\ g_{1,1}, g_{2,1}, \ldots, g_{n,1} \end{pmatrix}\right) \qquad \mathsf{t} := \begin{pmatrix} \alpha_{1,0}, \alpha_{2,0} \ldots, \alpha_{n,0} \\ \alpha_{1,1}, \alpha_{2,1}, \ldots, \alpha_{n,1} \end{pmatrix}. \tag{2}$$

- $\mathsf{H}(\mathsf{k}, \mathsf{x}; r)$: Parse k as in Equation (2), sample $r \xleftarrow{\$} \text{Sample}(\mathbb{G})$, set $\mathsf{h} := g^r \cdot \prod_{j\in[n]} g_{j,\mathsf{x}_j}$ and output h
- $\mathsf{H}^{-1}(\mathsf{t}, (\mathsf{x}, r), \mathsf{x}')$: Parse t as in Equation (2), compute $r' := r + \sum_{j\in[n]}(\alpha_{j,\mathsf{x}_j} - \alpha_{j,\mathsf{x}'_j}) \mod p$. Output r'.
- $\text{Enc}(\mathsf{k}, (\mathsf{h}, i, b), \mathsf{m})$: Parse k as in Equation (2), $\mathsf{h} \in \mathbb{G}$ and $\mathsf{m} \in \{0,1\}$. Sample $\rho \xleftarrow{\$} \text{Sample}(\mathbb{G})$ and proceed as follows:
  (1) Set $c := g^\rho$ and $c' := \mathsf{h}^\rho$.
  (2) For every $j \in [n]\backslash\{i\}$, set $c_{j,0} := g_{j,0}^\rho$ and $c_{j,1} := g_{j,1}^\rho$.
  (3) Set $c_{i,0} := \perp$ and $c_{i,1} := \perp$.
  (4) Set $e := \mathsf{m} \oplus \text{HardCore}(g_{i,b}^\rho)$.[18]
  (5) Output $\mathsf{ct} := (e, c, c', \begin{pmatrix} c_{1,0}, c_{2,0} \ldots, c_{n,0} \\ c_{1,1}, c_{2,1}, \ldots, c_{n,1} \end{pmatrix})$.
- $\text{Dec}(\mathsf{k}, (\mathsf{x}, r), \mathsf{ct})$: Parse $\mathsf{ct} = (e, c, c', \begin{pmatrix} c_{1,0}, c_{2,0} \ldots, c_{n,0} \\ c_{1,1}, c_{2,1}, \ldots, c_{n,1} \end{pmatrix})$
  Output $e \oplus \text{HardCore}(\frac{c'}{c^r \cdot \prod_{j\in[n]\backslash\{i\}} c_{j,\mathsf{x}_j}})$.

Multi-bit Encryption. The encryption procedure described above encrypts single bit messages. Longer messages can be encrypted by encrypting individual bits.

Lemma 5.1. *Assuming that* CDH($\mathbb{G}$) *is hard, the construction described above is a chameleon encryption scheme, i.e., it satisfies Definition 4.1.*

Proof. We need to argue the trapdoor collision property, uniformity property, correctness of encryption property, and semantic security of the scheme above.

- **Uniformity:** Observe that for all k and x, we have that $\mathsf{H}(\mathsf{k}, \mathsf{x}; r) = g^r \cdot \prod_{j\in[n]} g_{j,\mathsf{x}_j}$ is statistically close to a uniform element in $\mathbb{G}$. This is because r is sampled statistically close to uniform in $\mathbb{Z}_p$, where p is the order of $\mathbb{G}$.
- **Trapdoor Collisions:** For any choice of $\mathsf{x}, \mathsf{x}', r, \mathsf{k}, \mathsf{t}$, the value r' is obtained as $r + \sum_{j\in[n]}(\alpha_{j,\mathsf{x}_j} - \alpha_{j,\mathsf{x}'_j}) \mod p$. We need to show that $\mathsf{H}(\mathsf{k}, \mathsf{x}'; r')$ is equal to $\mathsf{H}(\mathsf{k}, \mathsf{x}; r)$. This

[16] We will later discuss instantiations of $\mathbb{G}$ that are of prime order and composite order. The use of Sample($\mathbb{G}$) procedure is done to unify these two instantiations.
[17] We also implicitly include the public and secret parameters for the group $\mathbb{G}$ in k and t, respectively.
[18] We assume that the HardCore(g^{ab}) is a hardcore bit of g^{ab} given g^a and g^b. If a deterministic hard-core bit for the specific function is not known, then we can always use the Goldreich and Levin [1989] construction.

can be established as follows:

$$\begin{aligned}
\mathsf{H}(\mathsf{k}, \mathsf{x}'; r') &= g^{r'} \cdot \prod_{j\in[n]} g_{j,\mathsf{x}'_j} \\
&= g^{r+\sum_{j\in[n]}(\alpha_{j,\mathsf{x}_j}-\alpha_{j,\mathsf{x}'_j})} \cdot \prod_{j\in[n]} g^{\alpha_{j,\mathsf{x}'_j}} \\
&= g^{r+\sum_{j\in[n]}(\alpha_{j,\mathsf{x}_j}-\alpha_{j,\mathsf{x}'_j})} \cdot g^{\sum_{j\in[n]}\alpha_{j,\mathsf{x}'_j}} \\
&= g^{r+\sum_{j\in[n]}\alpha_{j,\mathsf{x}_j}} \\
&= g^{r} \cdot \prod_{j\in[n]} g^{\alpha_{j,\mathsf{x}_j}} \\
&= g^{r} \cdot \prod_{j\in[n]} g_{j,\mathsf{x}_j} \\
&= \mathsf{H}(\mathsf{k}, \mathsf{x}; r).
\end{aligned}$$

Moreover, as r is statistically close to uniform in $\mathbb{Z}_p$, $r' := r + \sum_{j\in[n]}(\alpha_{j,\mathsf{x}_j} - \alpha_{j,\mathsf{x}'_j}) \mod p$ is also statistically close to uniform in $\mathbb{Z}_p$.

- **Correctness:** For any choice of $\mathsf{x} \in \{0,1\}^n$, coins r, index $i \in [n]$, and message $\mathsf{m} \in \{0,1\}$ if $(\mathsf{k}, \mathsf{t}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, n)$, $\mathsf{h} := \mathsf{H}(\mathsf{k}, \mathsf{x}; r)$, and $\mathsf{ct} := \mathsf{Enc}(\mathsf{k}, (\mathsf{h}, i, \mathsf{x}_i), \mathsf{m})$, then we have:

$$\begin{aligned}
\frac{c'}{c^r \cdot \prod_{j\in[n]\setminus\{i\}} c_{j,\mathsf{x}_j}} &= \frac{\mathsf{h}^\rho}{g^{\rho\cdot r} \cdot \prod_{j\in[n]\setminus\{i\}} g^{\rho\cdot\alpha_{j,\mathsf{x}_j}}} \\
&= \frac{g^{\rho\cdot r}\prod_{j\in[n]} g^{\rho\cdot\alpha_{j,\mathsf{x}_j}}}{g^{\rho\cdot r} \cdot \prod_{j\in[n]\setminus\{i\}} g^{\rho\cdot\alpha_{j,\mathsf{x}_j}}} \\
&= g^{\rho\cdot\alpha_{i,\mathsf{x}_i}} \\
&= g^{\rho}_{i,\mathsf{x}_i}.
\end{aligned}$$

Using the above calculation and parsing $\mathsf{ct} = (e, c, c', \binom{c_{1,0}, c_{2,0} \ldots, c_{n,0}}{c_{1,1}, c_{2,1}, \ldots, c_{n,1}})$ allows us to conclude that

$$\begin{aligned}
\mathsf{Dec}\,(\mathsf{k}, (\mathsf{x}, r), \mathsf{ct}) &= e \oplus \mathsf{HardCore}\left(\frac{c'}{c^r \cdot \prod_{j\in[n]\setminus\{i\}} c_{j,\mathsf{x}_j}}\right) \\
&= e \oplus \mathsf{HardCore}(g^{\rho}_{i,\mathsf{x}_i}) \\
&= \mathsf{m} \oplus \mathsf{HardCore}(g^{\rho}_{i,\mathsf{x}_i}) \oplus \mathsf{HardCore}(g^{\rho}_{i,\mathsf{x}_i}) \\
&= \mathsf{m}.
\end{aligned}$$

- **Security:** For the sake of contradiction, let us assume that there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and a non-negligible function $\mu(\cdot)$ such that

$$\Pr[\mathsf{IND}^{\mathsf{CE}}_{\mathcal{A}}(1^\lambda) = 1] \geq \frac{1}{2} + \mu(\lambda).$$

Now, we will provide a PPT reduction $\mathcal{R}^{\mathcal{A}}$ that on input $g, U = g^u, V = g^v$ correctly computes the hardcore bit $\mathsf{HardCore}(g^{uv})$ with probability $\frac{1}{2} + \nu(\lambda)$ for some non-negligible function ν. Formally, the reduction $\mathcal{R}^{\mathcal{A}}(g, U, V)$ proceeds as follows:

(1) For each $j \in [n]$, sample $\alpha_{j,0}, \alpha_{j,1} \xleftarrow{\$} \mathsf{Sample}(\mathbb{G})$ and set $g_{j,0} := g^{\alpha_{j,0}}$ and $g_{j,1} := g^{\alpha_{j,1}}$.

(2) Sample $z \xleftarrow{\$} \{0,1\}$ and $i^* \xleftarrow{\$} [n]$ and reassign $g_{i^*,z} := U$. Finally, set

$$\mathsf{k} := \left(g, \begin{pmatrix} g_{1,0}, g_{2,0} \ldots, g_{n,0} \\ g_{1,1}, g_{2,1}, \ldots, g_{n,1} \end{pmatrix}\right).$$

(3) $(\mathsf{x}, r, i) \xleftarrow{\$} \mathcal{A}_1(\mathsf{k})$.

(4) If $i \neq i^*$ or $\mathsf{x}_i = z$, then skip rest of the steps and output a random bit $b \xleftarrow{\$} \{0,1\}$.

(5) Otherwise, set $\mathsf{h} := \mathsf{H}(\mathsf{k}, \mathsf{x}; r)$ and $\mathsf{ct} := (e, c, c', \begin{pmatrix} c_{1,0}, c_{2,0} \ldots, c_{n,0} \\ c_{1,1}, c_{2,1}, \ldots, c_{n,1} \end{pmatrix})$ where:

$$\begin{aligned} & c := V && c' := V^{r+\sum_{j\in[n]} \alpha_{i,\mathsf{x}_i}}, \\ \forall j \in [n]\backslash\{i\}, \quad & c_{j,0} := V^{\alpha_{j,0}} && c_{j,1} := V^{\alpha_{j,1}}, \\ & e \xleftarrow{\$} \{0,1\}. \end{aligned}$$

(6) $b \xleftarrow{\$} \mathcal{A}_2(\mathsf{k}, (\mathsf{x}, r), \mathsf{ct})$.

(7) Output $b \oplus e$.

Let E be the event that the $i = i^*$ and $x_i \neq z$. Now observe that the distribution of k in Step 3 is identical to the distribution resulting from Gen. This implies that (1) the view of the attacker in Step 3 is identical to experiment $\mathsf{IND}^{\mathsf{CE}}_{\mathcal{A}}$, and (2) $\Pr[E]$ is close to $\frac{1}{2n}$ up to a negligible additive term. Furthermore, conditioned on the fact that E occurs, we have that the view of the attacker in Step 3 is statistically close to experiment $\mathsf{IND}^{\mathsf{CE}}_{\mathcal{A}}$ where ct is an encryption of $e \oplus \mathsf{HardCore}(g^{uv})$ (where $U = g^u$ and $V = g^v$). Now, if $\mathcal{A}_2$ in Step 6 correctly predicts $e \oplus \mathsf{HardCore}(g^{uv})$, then we have that the output of our reduction $\mathcal{R}$ is a correct prediction of $\mathsf{HardCore}(g^{uv})$. Thus, we conclude that $\mathcal{R}$ predicts $\mathsf{HardCore}(g^{uv})$ correctly with probability at least $\frac{1}{2} \cdot (1 - \frac{1}{2n}) + \frac{1}{2n} \cdot (\frac{1}{2} + \mu) = \frac{1}{2} + \frac{\mu}{2n}$ up to a negligible additive term. □

5.1 Instantiations

Instantiating by prime order groups. Our scheme can be directly instantiated in any prime order group $\mathbb{G}$ in which the computational Diffie-Hellman problem is hard.

Instantiating by composite order groups and reduction to the Factoring Assumption. Consider the group of quadratic residues $\mathbb{QR}_N$ over a Blum integer $N = PQ$ (P and Q are large safe primes[19] with $P \equiv Q \equiv 3 \pmod 4$). Let g be a random generator of $\mathbb{G}$ and $\mathsf{Sample}(\mathbb{G})$ outputs a uniformly random number from the set $[(N-1)/4]$. Shmuely [1985] and McCurley [1988] proved that the $\mathsf{CDH}(\mathbb{QR}_N)$ problem is at least as hard as FACT (also see Biham et al. [1997] and Hofheinz and Kiltz [2009]).

For this instantiation, we assume that the Gen algorithm generates a fresh Blum integer $N = PQ$, includes N in the public key k, and $|\mathbb{G}| = |\mathbb{QR}_N| = \phi(N)/4 = pq$ in the trapdoor t. Notice that only the trapdoor-collision algorithm H^{-1} needs to know the group-order $|\mathbb{G}| = pq$, while all other algorithms use the public sampling algorithm $\mathsf{Sample}(\mathbb{G})$.

6 CONSTRUCTION OF IDENTITY-BASED ENCRYPTION

In this section, we describe our construction of IBE from chameleon encryption. Let $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^{\leq n} \cup \{\varepsilon\} \to \{0,1\}^\lambda$ be a pseudorandom function, $(\mathsf{Gen}, \mathsf{H}, \mathsf{H}^{-1}, \mathsf{Enc}, \mathsf{Dec})$ be a chameleon

[19] A prime number $P > 2$ is called safe prime if $(P-1)/2$ is also prime.

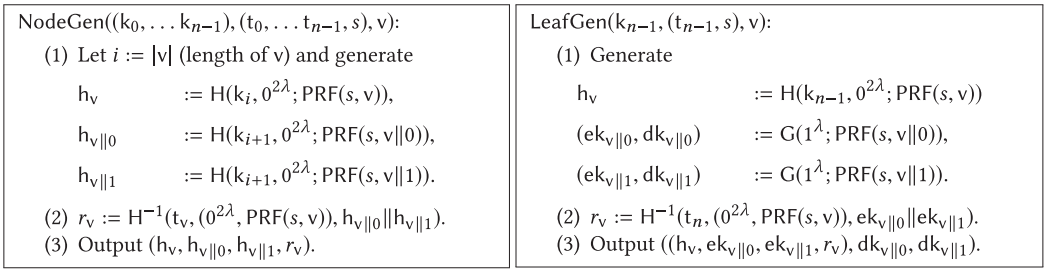

Fig. 7. Description of NodeGen and LeafGen.

encryption scheme, and (G, E, D) be any semantically secure public-key encryption scheme.[20] We let $\mathsf{ID}[i]$ denote the ith bit of ID and let $\mathsf{ID}[1 \dots i]$ denote the first i bits of ID. We define $\mathsf{ID}[1 \dots 0]$ as the empty string denoted by ε.

The NodeGen and LeafGen functions. As explained in the introduction, we need an exponentially sized tree of hash values. The functions NodeGen and LeafGen provide efficient access to the *hash value* corresponding to any node in this (exponential sized) tree. We will use these functions repeatedly in our construction. The NodeGen function takes as input the hash keys $\mathsf{k}_0, \dots \mathsf{k}_{n-1}$ and corresponding trapdoors $\mathsf{t}_0, \dots \mathsf{t}_{n-1}$, the PRF seed s, and a node $\mathsf{v} \in \{0,1\}^{\leq n-2} \cup \{\varepsilon\}$. However, the LeafGen function takes as input the hash key k_{n-1} and corresponding trapdoor t_{n-1}, the PRF seed s, and a node $\mathsf{v} \in \{0,1\}^{n-1}$. The NodeGen and LeafGen functions are described in Figure 7.

Construction. We describe our IBE scheme (Setup, KeyGen, Encrypt, Decrypt).

- $\mathsf{Setup}(1^\lambda, 1^n)$: Proceed as[21] follows:
  (1) Sample $s \xleftarrow{\$} \{0,1\}^\lambda$ (seed for the pseudorandom function PRF).
  (2) For each $i \in \{0, \dots n-1\}$ sample $(\mathsf{k}_i, \mathsf{t}_i) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, 2\lambda)$.
  (3) Obtain $(\mathsf{h}_\varepsilon, \mathsf{h}_0, \mathsf{h}_1, r_\varepsilon) := \mathsf{NodeGen}((\mathsf{k}_0, \dots \mathsf{k}_{n-1}), (\mathsf{t}_0, \dots \mathsf{t}_{n-1}, s), \varepsilon)$.
  (4) Output $(\mathsf{mpk}, \mathsf{msk})$ where $\mathsf{mpk} := (\mathsf{k}_0, \dots \mathsf{k}_{n-1}, \mathsf{h}_\varepsilon)$ and $\mathsf{msk} := (\mathsf{mpk}, \mathsf{t}_0, \dots \mathsf{t}_{n-1}, s)$.
- $\mathsf{KeyGen}(\mathsf{msk} = ((\mathsf{k}_0, \dots \mathsf{k}_{n-1}, \mathsf{h}_\varepsilon), \mathsf{t}_0, \dots \mathsf{t}_{n-1}, s), \mathsf{ID} \in \{0,1\}^n)$:

  $V := \{\varepsilon, \mathsf{ID}[1], \dots \mathsf{ID}[1 \dots n-1]\}$, where ε is the emptystring.
  For all $\mathsf{v} \in V \backslash \{\mathsf{ID}[1 \dots n-1]\}$:
  $\quad \mathsf{lk}_\mathsf{v} := \mathsf{NodeGen}((\mathsf{k}_0, \dots \mathsf{k}_{n-1}), (\mathsf{t}_0, \dots \mathsf{t}_{n-1}, s), \mathsf{v})$.
  For $\mathsf{v} = \mathsf{ID}[1 \dots n-1]$, set $(\mathsf{lk}_\mathsf{v}, \mathsf{dk}_{\mathsf{v}||0}, \mathsf{dk}_{\mathsf{v}||1}) := \mathsf{LeafGen}(\mathsf{k}_{n-1}, (\mathsf{t}_{n-1}, s), \mathsf{v})$.
  $\mathsf{sk}_\mathsf{ID} := (\mathsf{ID}, \{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v} \in V}, \mathsf{dk}_\mathsf{ID})$.

- $\mathsf{Encrypt}(\mathsf{mpk} = (\mathsf{k}_0, \dots \mathsf{k}_{n-1}, \mathsf{h}_\varepsilon), \mathsf{ID} \in \{0,1\}^n, \mathsf{m})$: Before describing the encryption procedure, we describe two circuits[22] that will be garbled during the encryption process.

[20] The algorithm G takes as input the security parameter 1^λ and generates encryption key and decryption key pair ek and dk, respectively, where the encryption key ek is assumed to be λ bits long. The encryption algorithm E(ek, m) takes as input an encryption key ek and a message m and outputs a ciphertext ct. Finally, the decryption algorithm D(dk, ct) takes as input the secret key and the ciphertext and outputs the encrypted message m.

[21] The IBE scheme defined in Section 3 does not fix the length of identities that it can be used with. However, in this section, we fix the length of identities at setup time and use appropriately changed definitions. Looking ahead, the HIBE construction in Section 7 works for identities of arbitrary length.

[22] Random coins used by these circuits are hardwired in them. For simplicity, we do not mention them explicitly.

— $\mathsf{T}[\mathsf{m}](\mathsf{ek})$: Compute and output $\mathsf{E}(\mathsf{ek}, \mathsf{m})$.
— $\mathsf{P}[\beta \in \{0,1\}, \mathsf{k}, \overline{\mathsf{lab}}](\mathsf{h})$: Compute and output $\{\mathsf{Enc}(\mathsf{k}, (\mathsf{h}, j + \beta \cdot \lambda, b), \mathsf{lab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}$, where $\overline{\mathsf{lab}}$ is short for $\{\mathsf{lab}_{j,b}\}_{j\in[\lambda], b\in\{0,1\}}$.

Encryption proceeds as follows:

(1) Compute $\tilde{T}$ as:

$$(\tilde{T}, \overline{\mathsf{lab}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{T}[\mathsf{m}]).$$

(2) For $i = n-1, \ldots, 0$ generate $(\tilde{P}^i, \overline{\mathsf{lab}}') \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{P}[\mathsf{ID}[i+1], \mathsf{k}_i, \overline{\mathsf{lab}}])$ and set $\overline{\mathsf{lab}} := \overline{\mathsf{lab}}'$.

(3) Output $\mathsf{ct} := (\{\mathsf{lab}_{j,\mathsf{h}_{\varepsilon,j}}\}_{j\in[\lambda]}, \{\tilde{P}^0, \ldots, \tilde{P}^{n-1}, \tilde{T}\})$ where $\mathsf{h}_{\varepsilon,j}$ is the jth bit of h_ε.

- $\mathsf{Decrypt}(\mathsf{ct}, \mathsf{sk}_{\mathsf{ID}} = (\mathsf{ID}, \{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v}\in V}), \mathsf{dk}_{\mathsf{ID}})$: Decryption proceeds as follows:

(1) Parse ct as $(\{\mathsf{lab}_{j,\mathsf{h}_{\varepsilon,j}}\}_{j\in[\lambda]}, \{\tilde{P}^0, \ldots, \tilde{P}^{n-1}, \tilde{T}\})$.

(2) Parse lk_v as $(\mathsf{h}_\mathsf{v}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|1}, r_\mathsf{v})$ for each $\mathsf{v} \in V\backslash\{\mathsf{ID}[1\ldots n-1]\}$. (Recall $V = \{\varepsilon, \mathsf{ID}[1] \ldots \mathsf{ID}[1\ldots n-1]\}$.)

(3) And for $\mathsf{v} = \mathsf{ID}[1\ldots n-1]$, parse lk_v as $(\mathsf{h}_\mathsf{v}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1}, r_\mathsf{v})$.

(4) Set $y := \mathsf{h}_\varepsilon$.

(5) For each $i \in \{0, \ldots n-1\}$, set $\mathsf{v} := \mathsf{ID}[1\ldots i]$, and proceed as follows:

(a) $\{e_{j,b}\}_{j\in[\lambda], b\in\{0,1\}} := \mathsf{Eval}(\tilde{P}^i, \{\mathsf{lab}_{j,y_j}\}_{j\in[\lambda]})$.

(b) If $i = n-1$, then set $y := \mathsf{ek}_{\mathsf{ID}}$ and for each $j \in [\lambda]$, compute

$$\mathsf{lab}_{j,y_j} := \mathsf{Dec}(\mathsf{k}_\mathsf{v}, e_{j,y_j}, (\mathsf{ek}_{\mathsf{v}\|0} \| \mathsf{ek}_{\mathsf{v}\|1}, r_\mathsf{v})).$$

(c) If $i \neq n-1$, then set $y := \mathsf{h}_\mathsf{v}$ and for each $j \in [\lambda]$, compute

$$\mathsf{lab}_{j,y_j} := \mathsf{Dec}(\mathsf{k}_\mathsf{v}, e_{j,y_j}, (\mathsf{h}_{\mathsf{v}\|0} \| \mathsf{h}_{\mathsf{v}\|1}, r_\mathsf{v})).$$

(6) Compute $f := \mathsf{Eval}(\tilde{T}, \{\mathsf{lab}_{j,y_j}\}_{j\in[\lambda]})$.

(7) Output $\mathsf{m} := \mathsf{Dec}(\mathsf{dk}_{\mathsf{ID}}, f)$.

A note on efficiency. The most computationally intensive part of the construction is the non-black-box use of Enc inside garblings of the circuit P and E inside garbling of the circuit T. However, we note that not all of the computation corresponding to Enc and E needs to be performed inside the garbled circuit, and it might be possible to push some of it outside of the garbled circuits. In particular, when Enc is instantiated with the DDH based chameleon encryption scheme, then we can reduce each Enc to a single modular exponentiation inside the garbled circuit. Similar optimizations can be performed for E. In short, this reduces the number of non-black-box modular exponentiations to 2λ for every circuit P and 1 for the circuit T. Finally, we note that additional improvements in efficiency might be possible by increasing the arity of the tree from 2 to a larger value. This would also reduce the depth of the tree and thereby reduce the number of non-black-box modular exponentiations needed.

6.1 Proof of Correctness

We will first show that our scheme is correct. For any identity ID, let $V = \{\varepsilon, \mathsf{ID}[1], \ldots \mathsf{ID}[1\ldots n-1]\}$. Then the secret key $\mathsf{sk}_{\mathsf{ID}}$ consists of $(\mathsf{ID}, \{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v}\in V}, \mathsf{dk}_{\mathsf{ID}})$. We will argue that decryption of a correctly generated ciphertext reveals the original message. Note that by construction (and the trapdoor collision property of the chameleon encryption scheme for the first equation below) for all nodes $\mathsf{v} \in V\backslash\{\mathsf{ID}[1\ldots n-1]\}$, we have that:

$$\mathsf{H}(\mathsf{k}_{|\mathsf{v}|}, \mathsf{h}_{\mathsf{v}\|0} \| \mathsf{h}_{\mathsf{v}\|1}; r_\mathsf{v}) = \mathsf{h}_\mathsf{v},$$

and additionally for $\mathsf{v} = \mathsf{ID}[1 \dots n-1]$, we have

$$\mathsf{H}(\mathsf{k}_{n-1}, \mathsf{ek}_{\mathsf{v}\|0} \| \mathsf{ek}_{\mathsf{v}\|1}; r_{\mathsf{v}}) = \mathsf{h}_{\mathsf{v}}.$$

Next consider a ciphertext $\mathsf{ct} = (\{\mathsf{lab}_{j,\mathsf{h}_{\varepsilon,j}}\}_{j\in[\lambda]}, \{\tilde{P}^0, \dots, \tilde{P}^{n-1}, \tilde{T}\})$. We argue correctness as each step of decryption is performed. By correctness of garbled circuits, we have that the evaluation of $\tilde{P}^0$ yields correctly formed ciphertexts $e_{j,b}$ that are encryptions of labels of the next garbled circuit $\tilde{P}^1$. Next, by correctness of Dec of the chameleon encryption scheme, we have that the decrypting of the appropriate ciphertexts yields the correct labels $\{\mathsf{lab}_{j,\mathsf{h}_{\mathsf{ID}[1],j}}\}_{j\in[\lambda]}$ for the next garbled circuit, namely, $\tilde{P}^1$. Following the same argument, we can argue that the decryption of the appropriate ciphertexts generated by $\tilde{P}^1$ yields the correct input labels for $\tilde{P}^2$. Repeatedly applying this argument allows us to conclude that the last garbled circuit $\tilde{P}^{n-1}$ outputs labels corresponding to $\mathsf{ek}_{\mathsf{ID}}$ as input for the circuit T, which outputs an encryption of m under $\mathsf{ek}_{\mathsf{ID}}$. Finally, using the correctness of the public-key encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$, we have that the recovered message m is the same as the one encrypted.

6.2 Proof of Security

Theorem 6.1. *Let* $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^{\leq n} \cup \{\varepsilon\} \to \{0,1\}^\lambda$ *be a pseudorandom function,* $(\mathsf{Gen}, \mathsf{H}, \mathsf{H}^{-1}, \mathsf{Enc}, \mathsf{Dec})$ *be a secure chameleon encryption scheme, and* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *be a semantically secure public-key encryption scheme. Then* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is a fully secure IBE scheme.*

Proof. For the sake of contradiction, we proceed by assuming that there exists an adversary $\mathcal{A}$ such that $\Pr[\mathsf{IND}^{\mathsf{IBE}}_{\mathcal{A}}(1^\lambda) = 1] \geq \frac{1}{2} + \epsilon$ for a non-negligible ϵ (in λ), where $\mathsf{IND}^{\mathsf{IBE}}_{\mathcal{A}}$ is shown in Figure 4. Assume further that q is a polynomial upper bound for the running-time of $\mathcal{A}$, and thus also an upper bound for the number of $\mathcal{A}$'s key queries. Security follows by a sequence of hybrids. In our hybrids, changes are made in how the secret key queries of the adversary $\mathcal{A}$ are answered and how the challenge ciphertext is generated. Furthermore, these changes are intertwined and need to be done carefully. Our proof consists of a sequence of $n+2$ hybrids $\mathcal{H}_{-1}, \mathcal{H}_0, \mathcal{H}_1, \dots \mathcal{H}_{n+1}$. We next describe these hybrids.

- $\mathcal{H}_{-1}$: This hybrid corresponds to the experiment $\mathsf{IND}^{\mathsf{IBE}}_{\mathcal{A}}$, as shown in Figure 4.
- $\mathcal{H}_0$: In this hybrid, we change how the public parameters are generated and how the adversary's requests to the KeyGen oracle are answered. Specifically, we replace all pseudorandom function calls $\mathsf{PRF}(s, \cdot)$ with a random function.

  The only change from $\mathcal{H}_{-1}$ to $\mathcal{H}_0$ is that calls to a pseudorandom are replaced by a random function. Therefore, the indistinguishability between the two hybrids follows directly from the pseudorandomness property of the pseudorandom function.
- $\mathcal{H}_\tau$ for $\tau \in \{0 \dots n\}$: For every τ, this hybrid is identical to the experiment $\mathcal{H}_0$ except in how the ciphertext is generated. Recall that the challenge ciphertext consists of a sequence of $n+1$ garbled circuits. In hybrid $\mathcal{H}_\tau$, we generate the first τ of these garbled circuits using the simulator provided by the garbled circuit construction. The outputs hard-coded in the simulated circuits are set to be consistent with the output that would have resulted from the execution of honestly generated garbled circuits. More formally, for the challenge identity ID^* the challenge ciphertext is generated as follows (modifications with respect to honest ciphertext generation have been highlighted in red). Even though the adversary never queries $\mathsf{sk}_{\mathsf{ID}}$, we can generate it locally. In particular, it contains the values $\mathsf{lk}_{\mathsf{v}} = (\mathsf{h}_{\mathsf{v}}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|1}, r_{\mathsf{v}})$ for each $\mathsf{v} \in \{\varepsilon, \dots \mathsf{ID}[1 \dots n-2]\}$, $\mathsf{lk}_{\mathsf{v}} = (\mathsf{h}_{\mathsf{v}}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1}, r_{\mathsf{v}})$ for each $\mathsf{v} = \mathsf{ID}[1 \dots n-1]$, and $\mathsf{dk}_{\mathsf{ID}^*}$.

(1) Compute $\tilde{T}$ as:
If $\tau \neq n$

$$(\tilde{T}, \overline{\mathsf{lab}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{T}[\mathsf{m}]),$$

where $\overline{\mathsf{lab}} = \{\mathsf{lab}_{j,b}\}_{j\in[\lambda], b\in\{0,1\}}$. Else set $y = \mathsf{ek}_{\mathsf{ID}^*}$ and generate garbled circuit as,

$$(\tilde{T}, \{\mathsf{lab}_{j,y_j}\}_{j\in[\lambda]}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{E}(y, \mathsf{m}))$$

and set $\overline{\mathsf{lab}} := \{\mathsf{lab}_{j,y_j}, \mathsf{lab}_{j,y_j}\}_{j\in[\lambda]}$.

(2) For $i = n-1, \ldots, \tau$ generate $(\tilde{P}^i, \overline{\mathsf{lab}}') \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{P}[\mathsf{ID}[i+1], \mathsf{k}_i, \overline{\mathsf{lab}}])$ and set $\overline{\mathsf{lab}} := \overline{\mathsf{lab}}'$.

(3) For $i = \tau - 1, \ldots, 0$, *set* $\mathsf{v} = \mathsf{ID}^*[1 \ldots i-1]$ and generate

$$\tilde{P}^i, \{\mathsf{lab}'_{j,\mathsf{h}_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k}_\mathsf{v}, (\mathsf{h}_\mathsf{v}, j, b), \mathsf{lab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}})$$

and set $\overline{\mathsf{lab}} := \{\mathsf{lab}'_{j,\mathsf{h}_{\mathsf{v},j}}, \mathsf{lab}'_{j,\mathsf{h}_{\mathsf{v},j}}\}_{j\in[\lambda]}$.

(4) Output $\mathsf{ct} := (\{\mathsf{lab}_{j,\mathsf{h}_{\varepsilon,j}}\}_{j\in[\lambda]}, \{\tilde{P}^0, \ldots, \tilde{P}^{n-1}, \tilde{T}\})$ where $\mathsf{h}_{\varepsilon,j}$ is the jth bit of h_ε.

The computational indistinguishability between hybrids $\mathcal{H}_{\tau-1}$ and $\mathcal{H}_\tau$ is based on Lemma 6.2, which is proved in Section 6.3.

Lemma 6.2. *For each $\tau \in \{1 \ldots n\}$ it is the case that $\mathcal{H}_{\tau-1} \overset{c}{\approx} \mathcal{H}_\tau$.*

- $\mathcal{H}_{n+1}$: This hybrid is same as $\mathcal{H}_n$ except that we change the ciphertext $\mathsf{E}(\mathsf{ek}_{\mathsf{ID}^*}, \mathsf{m})$ hardwired in the simulated garbling of the circuit T to be $\mathsf{E}(\mathsf{ek}_{\mathsf{ID}^*}, 0)$.

  We can use an adversary distinguishing between $\mathcal{H}_n$ and $\mathcal{H}_{n+1}$ to construct an attacker against the semantic security of the public-key encryption scheme (G, E, D) in the following way: Let q be a polynomial upper bound for the number of queries of the adversary (key-queries and the challenge query). The reduction gets as input a public key ek. There are two cases that might happen. In the first one, the challenge identity will be the *sibling* of an identity for which the adversary makes a key-query, i.e., the adversary gets to see $\mathsf{ek}_{\mathsf{v}\|0}$ and $\mathsf{ek}_{\mathsf{v}\|1}$ such that *wlog* $\mathsf{ek}_{\mathsf{v}\|0}$ is the leaf key for a key-query of the adversary and $\mathsf{ek}_{\mathsf{v}\|1}$ is the key for the challenge identity $\mathsf{v}\|1$. In the second one, this is not the case. To deal with both cases, the reduction first guesses an index $j^* \in \{0, \ldots, q\}$. We interpret the guess $j^* = 0$ as the second case, i.e., the challenge identity ID^* will not be the sibling of an identity for which the adversary makes a key query. All other guesses are interpreted as the challenge identity will be the sibling of the identity of the j^*-th key query. Therefore, if $j^* = 0$, then the reduction will not use ek before the adversary provides the challenge identity and then set $\mathsf{ek}_{\mathsf{ID}^*} = \mathsf{ek}$. In the second case the reduction will use ek as the leaf key of the sibling of the identity of the j^*-th key query. Once the adversary provides the challenge identity ID^*, the reduction first checks if its guess was correct. If not, then it aborts and outputs a random bit. Otherwise, if its guess turns out to be correct, which happens with probability at least $1/q$, then it forwards the challenge messages of $\mathcal{A}$ to the IND-CPA experiment, uses the challenge ciphertext in its own experiment, and outputs whatever the adversary outputs. Note that the adversary $\mathcal{A}$ never queries for $\mathsf{sk}_{\mathsf{ID}^*}$. Therefore, it is never provided the value $\mathsf{dk}_{\mathsf{ID}^*}$.

  It follows routinely that the advantage of the reduction is at least $1/q$ times the advantage of the adversary $\mathcal{A}$. This allows us to conclude that $\mathcal{H}_n \overset{c}{\approx} \mathcal{H}_{n+1}$.

  Finally, note that the hybrid $\mathcal{H}_{n+1}$ is information theoretically independent of the plaintext message m.

6.3 Proof of Lemma 6.2

The proof follows by a sequence of sub-hybrids $\mathcal{H}_{\tau,0}$ to $\mathcal{H}_{\tau,6}$ where $\mathcal{H}_{\tau,0}$ is same as $\mathcal{H}_{\tau-1}$ and $\mathcal{H}_{\tau,6}$ is same as $\mathcal{H}_{\tau}$.

- $\mathcal{H}_{\tau,0}$: This hybrid is same as $\mathcal{H}_{\tau-1}$.
- $\mathcal{H}_{\tau,1}$: Skip this hybrid if $\tau = n$. Otherwise, this hybrid is identical to $\mathcal{H}_{\tau,0}$, except that we change how the values $\mathsf{h_v}$ and r_v for $\mathsf{v} \in \{0,1\}^\tau$ (if needed to answer a KeyGen query of the adversary) are generated.
  Recall that in hybrid $\mathcal{H}_{\tau,0}$, $\mathsf{h_v}$ is generated as $\mathsf{H}(\mathsf{k}_\tau, 0^{2\lambda}; \omega_\mathsf{v})$ and then
  $$r_\mathsf{v} := \begin{cases} \mathsf{H}^{-1}(\mathsf{k}_\tau, (0^{2\lambda}, \omega_\mathsf{v}), \mathsf{h}_{\mathsf{v}\|0}\|\mathsf{h}_{\mathsf{v}\|1}) & \text{if } \tau < n-1 \\ \mathsf{H}^{-1}(\mathsf{k}_\tau, (0^{2\lambda}, \omega_\mathsf{v}), \mathsf{ek}_{\mathsf{v}\|0}\|\mathsf{ek}_{\mathsf{v}\|1}) & \text{otherwise} \end{cases}.$$
  In this hybrid, we generate r_v first as being chosen uniformly. Next,
  $$\mathsf{h_v} := \begin{cases} \mathsf{H}(\mathsf{k}_\tau, \mathsf{h}_{\mathsf{v}\|0}\|\mathsf{h}_{\mathsf{v}\|1}; r_\mathsf{v}) & \text{if } \tau < n-1 \\ \mathsf{H}(\mathsf{k}_\tau, \mathsf{ek}_{\mathsf{v}\|0}\|\mathsf{ek}_{\mathsf{v}\|1}; r_\mathsf{v}) & \text{otherwise} \end{cases}.$$
  Statistical indistinguishability of hybrids $\mathcal{H}_{\tau,0}$ and $\mathcal{H}_{\tau,1}$ follows from the trapdoor collision and uniformity properties of the chameleon encryption scheme.
- $\mathcal{H}_{\tau,2}$: We start with the case when $\tau < n$. For this case, in this hybrid, we change how the garbled circuit $\tilde{P}^\tau$ is generated. Let $\mathsf{v} = \mathsf{ID}^*[1 \ldots \tau]$ and recall that
  $$\mathsf{lk_v} = \begin{cases} (\mathsf{h_v}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|1}, r_\mathsf{v}) & \text{if } \tau < n-1 \\ (\mathsf{h_v}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1}, r_\mathsf{v}) & \text{if } \tau = n-1 \end{cases}.$$
  In this hybrid, we change the generation process of the garbled circuit $\tilde{P}^\tau$ from
  $$(\tilde{P}^\tau, \overline{\mathsf{lab}}') \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{P}[\mathsf{ID}[\tau+1], \mathsf{k}_\tau, \overline{\mathsf{lab}}])$$
  and setting $\overline{\mathsf{lab}} := \overline{\mathsf{lab}}'$ to
  $$\tilde{P}^i, \{\mathsf{lab}'_{j,\mathsf{h}_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k_v}, (\mathsf{h_v}, j, b), \mathsf{lab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}})$$
  and set $\overline{\mathsf{lab}} := \{\mathsf{lab}'_{j,\mathsf{h}_{\mathsf{v},j}}, \mathsf{lab}'_{j,\mathsf{h}_{\mathsf{v},j}}\}_{j\in[\lambda]}$.
  For the case when $\tau = n$, then we change computation of $\tilde{T}$ from
  $$(\tilde{T}, \overline{\mathsf{lab}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{T}[\mathsf{m}]),$$
  where $\overline{\mathsf{lab}} = \{\mathsf{lab}_{j,b}\}_{j\in[\lambda], b\in\{0,1\}}$ to setting $y = \mathsf{ek}_{\mathsf{ID}^*}$ and generating garbled circuit as,
  $$(\tilde{T}, \{\mathsf{lab}_{j,y_j}\}_{j\in[\lambda]}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{E}(y, \mathsf{m}))$$
  and setting $\overline{\mathsf{lab}} := \{\mathsf{lab}_{j,y_j}, \mathsf{lab}_{j,y_j}\}_{j\in[\lambda]}$.
  For the case when $\tau < n$, computational indistinguishability of hybrids $\mathcal{H}_{\tau,1}$ and $\mathcal{H}_{\tau,2}$ follows by the security of the garbling scheme and the fact that $\{\mathsf{Enc}(\mathsf{k_v}, (\mathsf{h_v}, j, b), \mathsf{lab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}$ is exactly the output of the circuit $\mathsf{P}[\mathsf{ID}[\tau+1], \mathsf{k}_\tau, \overline{\mathsf{lab}}]$ on input $\mathsf{h_v}$. However, for the case when $\tau = n$, then again indistinguishability of hybrids $\mathcal{H}_{n,1}$ and $\mathcal{H}_{n,2}$ follows by the security of the garbling scheme and the fact that $\mathsf{E}(\mathsf{ek}_{\mathsf{ID}^*}, \mathsf{m})$ is the output of the circuit $\mathsf{T}[\mathsf{m}]$ on input $\mathsf{ek}_{\mathsf{ID}^*}$.
- $\mathcal{H}_{\tau,3}$: Skip this hybrid if $\tau = n$. This hybrid is identical to $\mathcal{H}_{\tau,2}$, except that using $\mathsf{v} := \mathsf{ID}[1 \ldots \tau]$, we change
  $$\tilde{P}^i, \{\mathsf{lab}'_{j,\mathsf{h}_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k_v}, (\mathsf{h_v}, j, b), \mathsf{lab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}})$$

to

$$\tilde{P}^i, \{\mathsf{lab}'_{j,\mathsf{h}_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k}_\mathsf{v}, (\mathsf{h}_\mathsf{v}, j, b), \mathsf{lab}_{j,\mathsf{h}_{\mathsf{ID}[1\ldots\tau+1],j}})\}_{j\in[\lambda],b\in\{0,1\}}).$$

Notice that the trapdoor t_v is not used in this experiment. Therefore, computational indistinguishability of hybrids $\mathcal{H}_{\tau,2}$ and $\mathcal{H}_{\tau,3}$ follows by λ^2 invocations (one invocation for each bit of the λ labels) of the security of the chameleon encryption scheme. We now provide the reduction for one change below.

We outline a reduction to the security of the chameleon hash function. Specifically, the challenger provides a hash key k^* and the reduction needs to submit x^*, r^*. Recall that q is an upper bound for the number of queries by the adversary (including key and challenge queries). The reduction first guesses an index $j^* \in \{0,\ldots,q\}$, such that the node v^* on level τ of the j^*-th query is also on the root-to-leaf path of the challenge identity. We interpret $j^* = 0$ as the challenge identity not sharing a prefix of length τ with any keys queried by the adversary.

The reduction now sets $\mathsf{k}_\tau := \mathsf{k}^*$ and submits $\mathsf{x}^* := \mathsf{h}_{\mathsf{v}^*\|0}\|\mathsf{h}_{\mathsf{v}^*\|1}$ and randomly chosen coins $r_{\mathsf{v}^*} := r^*$ to the experiment once the label $\mathsf{h}_{\mathsf{v}^*} := \mathsf{H}(\mathsf{k}_\tau, \mathsf{x}^*; r^*)$ for the node v^* is needed in its simulation.

Once the adversary announces the challenge identity ID^*, the reduction checks if v^* is on the root-to-leaf path of ID^*; if not, it aborts and outputs a random bit. Clearly, it holds that v^* is on the root-to-leaf path of ID^* with probability $1/q$. Now, we can use the attackers ability to distinguish the encryptions of the provided labels to break the security of the chameleon encryption scheme, incurring a polynomial loss of $1/q$.

Remark. The ciphertexts hardwired inside the garbled circuit only provide the labels $\{\mathsf{lab}_{j,\mathsf{h}_{\mathsf{ID}[1\ldots\tau+1],j}}\}_{j\in[\lambda]}$ (in an information theoretical sense).

- $\mathcal{H}_{\tau,4}$: Skip this hybrid if $\tau = n$. In this hybrid, we undo the change made in going from hybrid $\mathcal{H}_{\tau,0}$ to hybrid $\mathcal{H}_{\tau,1}$, i.e., we go back to generating all h_v values using NodeGen and LeafGen. Statistical indistinguishability of hybrids $\mathcal{H}_{\tau,3}$ and $\mathcal{H}_{\tau,4}$ follows from the trapdoor collision and uniformity properties of the chameleon encryption scheme. Observe that the hybrid $\mathcal{H}_{\tau,4}$ is the same as hybrid $\mathcal{H}_\tau$.

7 CONSTRUCTION OF HIERARCHICAL IDENTITY-BASED ENCRYPTION

In this section, we describe our construction of HIBE from chameleon encryption. Let $(\mathsf{Gen}, \mathsf{H}, \mathsf{H}^{-1}, \mathsf{Enc}, \mathsf{Dec})$ be a chameleon encryption scheme, $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ be any semantically secure public-key encryption scheme, and $(\mathsf{F}, \mathsf{F.Delegate})$ be a delegatable pseudorandom function. We will assume that the output of F is sufficiently long for its purpose. We let $\mathsf{ID}[i]$ denote the ith bit of ID and $\mathsf{ID}[1\ldots i]$ denote the first i bits of ID (and $\mathsf{ID}[1\ldots 0] = \varepsilon$).

The NodeGen and NodeGen′ functions. As explained in the introduction, we need an unbounded tree of local-keys. The function NodeGen provides efficient access to *local-keys* corresponding to any node in this tree. We will use this function repeatedly in our construction. The function takes as input a global hash key k_G (a key of the chameleon hash function from $2\ell + 2\lambda$ bits to λ bits, where ℓ is specified later), a node $\mathsf{v} \in \{0,1\}^* \cup \{\varepsilon\}$ (ε denotes the empty string), and random coins (u, u_0, u_1). This function is explained in Figure 8.

We also define a function NodeGen′, which is identical to NodeGen except that it additionally takes a bit β as input and outputs $\mathsf{dk}_{\mathsf{v}\|\beta}$. More formally, $\mathsf{NodeGen}'(\mathsf{k}_G, \mathsf{v}, (s_1, s_2, s_3), \beta)$ behaves just like NodeGen but in Step 8 it outputs $\mathsf{dk}_{\mathsf{v}\|\beta}$.

$\mathsf{NodeGen}(\mathsf{k}_G, \mathsf{v}, (u, u_0, u_1))$:

(1) Parse u as $u = (\omega_1, \omega_2, \omega_3)$
(2) Generate $(\mathsf{k}_\mathsf{v}, \mathsf{t}_\mathsf{v}) := \mathsf{Gen}(1^\lambda, \lambda; \omega_1)$ and $\mathsf{h}_\mathsf{v} := \mathsf{H}(\mathsf{k}_\mathsf{v}, 0^\lambda; \omega_2)$.
(3) Analogous to the previous two steps generate $\mathsf{k}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|0}$ using coins u_0 and $\mathsf{k}_{\mathsf{v}\|1}, \mathsf{h}_{\mathsf{v}\|1}$ using coins u_1.
(4) Sample r'_v and generate $(\mathsf{ek}_{\mathsf{v}\|0}, \mathsf{dk}_{\mathsf{v}\|0}) \xleftarrow{\$} \mathsf{G}(1^\lambda)$ and $(\mathsf{ek}_{\mathsf{v}\|1}, \mathsf{dk}_{\mathsf{v}\|1}) \xleftarrow{\$} \mathsf{G}(1^\lambda)$ using ω_3 as random coins.
(5) $\mathsf{h}'_\mathsf{v} := \mathsf{H}(\mathsf{k}_G, \mathsf{k}_{\mathsf{v}\|0} || \mathsf{h}_{\mathsf{v}\|0} || \mathsf{k}_{\mathsf{v}\|1} || \mathsf{h}_{\mathsf{v}\|1} \| \mathsf{ek}_{\mathsf{v}\|0} \| \mathsf{ek}_{\mathsf{v}\|1}; r'_\mathsf{v})$.
(6) $r_\mathsf{v} := \mathsf{H}^{-1}(\mathsf{t}_\mathsf{v}, (0^\lambda, \omega_2), \mathsf{h}'_\mathsf{v})$.
(7) $\mathsf{lk}_\mathsf{v} := (\mathsf{k}_\mathsf{v}, \mathsf{h}_\mathsf{v}, r_\mathsf{v}, \mathsf{h}'_\mathsf{v}, r'_\mathsf{v}, \mathsf{k}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{k}_{\mathsf{v}\|1}, \mathsf{h}_{\mathsf{v}\|1}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1})$.
(8) Output lk_v

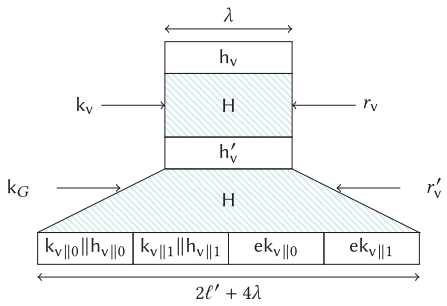

Fig. 8. Explanation on how NodeGen works. Strings ω_1, ω_2, and ω_3 are used as randomness for cryptographic functions and can be sufficiently expanded using a PRG.

Construction. We describe our HIBE scheme (Setup, KeyGen, Encrypt, Decrypt)

- $\mathsf{Setup}(1^\lambda)$: Proceed as follows:
  (1) Sample a seed $s \xleftarrow{\$} \{0,1\}^\lambda$ for the delegatable pseudorandom function F.
  (2) Set up a global hash function $(\mathsf{k}_G, \cdot) := \mathsf{Gen}(1^\lambda, 2\ell + 2\lambda)$[23] where $\ell = \ell' + \lambda$ and ℓ' is the length of a key k generated by $\mathsf{Gen}(1^\lambda, \lambda)$.
  (3) Compute $u := \mathsf{F}(s, \epsilon)$, $u_0 := \mathsf{F}(s, 0)$ and $u_1 := \mathsf{F}(s, 1)$.
  (4) Obtain $(\mathsf{k}_\varepsilon, \mathsf{h}_\varepsilon, r_\varepsilon, \mathsf{h}'_\varepsilon, r'_\varepsilon, \mathsf{k}_0, \mathsf{h}_0, \mathsf{k}_1, \mathsf{h}_1) := \mathsf{NodeGen}(\mathsf{k}_G, \varepsilon, (u, u_0, u_1))$.
  (5) Output $(\mathsf{mpk}, \mathsf{msk})$ where $\mathsf{mpk} := (\mathsf{k}_G, \mathsf{k}_\varepsilon, \mathsf{h}_\varepsilon)$ and $\mathsf{msk} = \mathsf{sk}_\varepsilon := (\varepsilon, \emptyset, s, \bot)$.
- $\mathsf{KeyGen}(\mathsf{sk}_\mathsf{ID} = (\mathsf{ID}, \{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v} \in V}, s_\mathsf{ID}, \mathsf{dk}_\mathsf{ID}), \mathsf{ID}' \in \{0,1\}^*)$:[24]

  Let $n := |\mathsf{ID}'|$ and set $V' := \{\mathsf{ID} \| \mathsf{ID}'[1 \ldots j-1]\}_{j \in [n]}$
  For all $\mathsf{v} \in V'$:
  $u := \mathsf{F}(s_\mathsf{ID}, \mathsf{v})$
  $u_0 := \mathsf{F}(s_\mathsf{ID}, \mathsf{v} \| 0)$
  $u_1 := \mathsf{F}(s_\mathsf{ID}, \mathsf{v} \| 1)$
  $\mathsf{lk}_\mathsf{v} := \mathsf{NodeGen}(\mathsf{k}_G, \mathsf{v}, (u, u_0, u_1))$
  Let $\mathsf{v} := \mathsf{ID} \| \mathsf{ID}'[1 \ldots n-1]$
  $u := \mathsf{F}(s_\mathsf{ID}, \mathsf{v})$
  $u_0 := \mathsf{F}(s_\mathsf{ID}, \mathsf{v} \| 0)$
  $u_1 := \mathsf{F}(s_\mathsf{ID}, \mathsf{v} \| 1)$
  $\mathsf{dk}_{\mathsf{ID}\|\mathsf{ID}'} := \mathsf{NodeGen}'(\mathsf{k}_G, \mathsf{v}, (u, u_0, u_1), \mathsf{ID}'[n])$
  $s_{\mathsf{ID}\|\mathsf{ID}'} := \mathsf{F.Delegate}(s_\mathsf{ID}, \mathsf{ID}')$
  Output $\mathsf{sk}_{\mathsf{ID}\|\mathsf{ID}'} := (\mathsf{ID}\|\mathsf{ID}', \{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v} \in V \cup V'}, s_{\mathsf{ID}\|\mathsf{ID}'}, \mathsf{dk}_{\mathsf{ID}\|\mathsf{ID}'})$

  Remark. We note that in our construction the secret key for any identity is unique regardless of how many iterations of KeyGen operations were performed to obtain it.
- $\mathsf{Encrypt}(\mathsf{mpk} = (\mathsf{k}_G, \mathsf{k}_\varepsilon, \mathsf{h}_\varepsilon), \mathsf{ID} \in \{0,1\}^n, \mathsf{m})$: Before describing the encryption procedure, we describe four circuits that will be garbled during the encryption process.
  $-\mathsf{T}[\mathsf{m}](\mathsf{ek})$: Compute and output $\mathsf{E}(\mathsf{ek}, \mathsf{m})$.

[23] The trapdoor for the global hash function is not needed in the construction or the proof and is therefore dropped.
[24] HIBE is often defined to have separate KeyGen and Delegate algorithms. For simplicity, we describe our scheme with just one KeyGen algorithm that enables both the tasks of decryption and delegation. Secret-keys without delegation capabilities can be obtained by dropping the third entry (the PRG seed) from sk_ID.

— $\mathsf{Q}_{last}[\beta \in \{0,1\}, \mathsf{k}_G, \overline{\mathsf{tlab}}](\mathsf{h})$: Compute and output $\{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}, j+\beta\cdot\lambda+2\ell, b), \mathsf{tlab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}$, where $\overline{\mathsf{tlab}}$ is short for $\{\mathsf{tlab}_{j,b}\}_{j\in[\lambda],b\in\{0,1\}}$.

— $\mathsf{Q}[\beta \in \{0,1\}, \mathsf{k}_G, \overline{\mathsf{plab}}](\mathsf{h})$: Compute and output $\{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}, j+\beta\cdot\ell, b), \mathsf{plab}_{j,b})\}_{j\in[\ell], b\in\{0,1\}}$, where $\overline{\mathsf{plab}}$ is short for $\{\mathsf{plab}_{j,b}\}_{j\in[\ell],b\in\{0,1\}}$.

— $\mathsf{P}[\overline{\mathsf{qlab}}](\mathsf{k},\mathsf{h})$: Compute and output $\{\mathsf{Enc}(\mathsf{k}, (\mathsf{h}, j, b), \mathsf{qlab}_{j,b})\}_{j\in[\lambda],b\in\{0,1\}}$, where $\overline{\mathsf{qlab}}$ is a shorthand for the set of labels $\{\mathsf{qlab}_{j,b}\}_{j\in[\lambda],b\in\{0,1\}}$.

Encryption proceeds as follows:

(1) Compute $\tilde{T}$ as:

$$(\tilde{T}, \overline{\mathsf{tlab}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, T[\mathsf{m}]).$$

(2) For $i = n, \ldots, 1$ generate

(a) If $i = n$, then

$$(\tilde{Q}^n, \overline{\mathsf{qlab}}^n) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{Q}_{last}[\mathsf{ID}[n], \mathsf{k}_G, \overline{\mathsf{tlab}}]),$$

else

$$(\tilde{Q}^i, \overline{\mathsf{qlab}}^i) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{Q}[\mathsf{ID}[i], \mathsf{k}_G, \overline{\mathsf{plab}}^{i+1}]).$$

(b) $(\tilde{P}^i, \overline{\mathsf{plab}}^i) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{P}[\overline{\mathsf{qlab}}^i])$.

(3) Set $x_\varepsilon := \mathsf{k}_\varepsilon \| \mathsf{h}_\varepsilon$.

(4) Output $\mathsf{ct} := (\{\mathsf{plab}^1_{j,x_{\varepsilon,j}}\}_{j\in[\ell]}, \{\tilde{P}^i, \tilde{Q}^i\}_{i\in[n]}, \tilde{T})$, where $x_{\varepsilon,j}$ is the jth bit of x_ε.

- $\mathsf{Decrypt}(\mathsf{ct}, \mathsf{sk}_{\mathsf{ID}} = (\mathsf{ID}, \{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v}\in V}, s_{\mathsf{ID}}, \mathsf{dk}_{\mathsf{ID}}))$: Decryption proceeds as follows:

(1) Parse ct as $(\{\mathsf{plab}^1_{j,x_{\varepsilon,j}}\}_{j\in[\ell]}, \{\tilde{P}^i, \tilde{Q}^i\}_{i\in[n]}, \tilde{T})$, where $x_\varepsilon := \mathsf{k}_\varepsilon \| \mathsf{h}_\varepsilon$ and $x_{\varepsilon,j}$ is its jth bit.

(2) Parse lk_v as $(\mathsf{h}_\mathsf{v}, r_\mathsf{v}, \mathsf{h}'_\mathsf{v}, r'_\mathsf{v}, \mathsf{k}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{k}_{\mathsf{v}\|1}, \mathsf{h}_{\mathsf{v}\|1}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1})$ for each $\mathsf{v} \in V$. (Recall that $V = \{\mathsf{ID}[1\ldots j-1]\}_{j\in[n]}$.)

(3) For each $i \in [n]$, proceed as follows:

(a) Set $\mathsf{v} := \mathsf{ID}[1\ldots i-1]$, $x_\mathsf{v} := \mathsf{k}_\mathsf{v} \| \mathsf{h}_\mathsf{v}$, $y_\mathsf{v} := \mathsf{h}'_\mathsf{v}$, and if $i < n$, then set $z_\mathsf{v} := \mathsf{k}_{\mathsf{v}\|\mathsf{ID}[i]} \| \mathsf{h}_{\mathsf{v}\|\mathsf{ID}[i]}$ else set $z_\mathsf{v} := \mathsf{ek}_{\mathsf{ID}}$.[25]

(b) $\{e^i_{j,b}\}_{j\in[\lambda],b\in\{0,1\}} := \mathsf{Eval}(\tilde{P}^i, \{\mathsf{plab}^i_{j,x_{\mathsf{v},j}}\}_{j\in[\ell]})$.

(c) For each $j \in [\lambda]$, compute $\mathsf{qlab}^i_{j,y_{\mathsf{v},j}} := \mathsf{Dec}(\mathsf{k}_\mathsf{v}, e^i_{j,y_{\mathsf{v},j}}, (\mathsf{h}'_\mathsf{v}, r_\mathsf{v}))$.

(d) If $i < n$, then,

$$\{f^i_{j,b}\}_{j\in[\ell],b\in\{0,1\}} := \mathsf{Eval}(\tilde{Q}^i, \mathsf{qlab}^i_{j,y_{\mathsf{v},j}})$$

and for each $j \in [\ell]$

$$\mathsf{plab}^{i+1}_{j,z_{\mathsf{v},j}} := \mathsf{Dec}(\mathsf{k}_G, f^i_{j,z_{\mathsf{v},j}}, (\mathsf{k}_{\mathsf{v}\|0}\|\mathsf{h}_{\mathsf{v}\|0}\|\mathsf{k}_{\mathsf{v}\|1}\|\mathsf{h}_{\mathsf{v}\|1}\|\mathsf{ek}_{\mathsf{v}\|0}\|\mathsf{ek}_{\mathsf{v}\|1}, r'_\mathsf{v}))$$

(e) else,

$$\{g_{j,b}\}_{j\in[\lambda],b\in\{0,1\}} := \mathsf{Eval}(\tilde{Q}^n, \mathsf{qlab}^n_{j,y_{\mathsf{v},j}})$$

and for each $j \in [\lambda]$

$$\mathsf{tlab}_{j,z_{\mathsf{v},j}} := \mathsf{Dec}(\mathsf{k}_G, g_{j,z_{\mathsf{v},j}}, (\mathsf{k}_{\mathsf{v}\|0}\|\mathsf{h}_{\mathsf{v}\|0}\|\mathsf{k}_{\mathsf{v}\|1}\|\mathsf{h}_{\mathsf{v}\|1}\|\mathsf{ek}_{\mathsf{v}\|0}\|\mathsf{ek}_{\mathsf{v}\|1}, r'_\mathsf{v})).$$

(4) Output $\mathsf{D}(\mathsf{dk}_{\mathsf{ID}}, \mathsf{Eval}(\tilde{T}, \{\mathsf{tlab}_{j,\mathsf{ek}_{\mathsf{ID},j}}\}_{j\in[\lambda]}))$.

[25] For $i < n$, z_v will become the x_v in next iteration.

7.1 Proof of Correctness

For any identity ID, let $V = \{\mathsf{ID}[1 \dots j-1]\}_{j \in [n]}$ be the set of nodes on the root-to-node path corresponding to identity ID. Then the secret key $\mathsf{sk}_{\mathsf{ID}}$ is of the form

$$\mathsf{sk}_{\mathsf{ID}} = (\mathsf{ID}, \{\mathsf{lk}_{\mathsf{v}}\}_{\mathsf{v} \in V}, s_{\mathsf{ID}}, \mathsf{dk}_{\mathsf{ID}}),$$

where s_{ID} is a delegated seed for the pseudorandom function F. Note that by construction, it holds via the trapdoor collision property of the chameleon encryption scheme that for all nodes $\mathsf{v} \in V$, we have:

$$\mathsf{H}(\mathsf{k}_G, \mathsf{k}_{\mathsf{v}\|0}||\mathsf{h}_{\mathsf{v}\|0}||\mathsf{k}_{\mathsf{v}\|1}||\mathsf{h}_{\mathsf{v}\|1}\|\mathsf{ek}_{\mathsf{v}\|0}\|\mathsf{ek}_{\mathsf{v}\|1}; r'_{\mathsf{v}}) = \mathsf{h}'_{\mathsf{v}},$$
$$\mathsf{H}(\mathsf{k}_{\mathsf{v}}, \mathsf{h}'_{\mathsf{v}}; r_{\mathsf{v}}) = \mathsf{h}_{\mathsf{v}}.$$

By correctness of garbled circuits, we have that the evaluation of $\tilde{P}^1$ yields correctly formed ciphertexts $f^1_{j,b}$. Next, by correctness of Dec of the chameleon encryption scheme, we have that the decrypted values $\mathsf{qlab}^1_{j,y_{\varepsilon,j}}$ are the correct input labels for the next garbled circuit $\tilde{Q}^1$. Following the same argument, we can argue that the decryption of ciphertexts generated by $\tilde{Q}^1$ yields the correct input labels for $\tilde{P}^2$. Repeatedly applying this argument allows us to conclude that the last garbled circuit $\tilde{Q}^n$ outputs correct encryptions of input labels of $\tilde{T}$. The decryption of appropriate ciphertexts among these and the execution of the garbled circuit $\tilde{T}$ using the obtained labels yields the ciphertext $\mathsf{E}(\mathsf{ek}_{\mathsf{ID}}, \mathsf{m})$, which can be decrypted using the decryption key $\mathsf{dk}_{\mathsf{ID}}$. Correctness of the last steps depends on the correctness of the public-key encryption scheme.

Next, the correctness of delegation follows from the fact that that for every ID and ID′

$$\mathsf{KeyGen}(\mathsf{sk}_{\varepsilon}, \mathsf{ID}\|\mathsf{ID}') = \mathsf{KeyGen}(\mathsf{KeyGen}(\mathsf{sk}_{\varepsilon}, \mathsf{ID}), \mathsf{ID}'),$$

which in turn follows directly from the the delegation property of the pseudorandom function F. Specifically, it holds for every ID and ID′ that

$$\mathsf{F.Delegate}(\mathsf{F.Delegate}(s, \mathsf{ID}), \mathsf{ID}') = \mathsf{F.Delegate}(s, \mathsf{ID}\|\mathsf{ID}')$$

and

$$\mathsf{F}(\mathsf{F.Delegate}(s, \mathsf{ID}), \mathsf{ID}') = \mathsf{F}(s, \mathsf{ID}\|\mathsf{ID}').$$

7.2 Proof of Security

Theorem 7.1. *Assume that* $(\mathsf{Gen}, \mathsf{H}, \mathsf{H}^{-1}, \mathsf{Enc}, \mathsf{Dec})$ *is a secure chameleon encryption scheme,* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *a semantically secure public-key encryption scheme, and* $(\mathsf{F}, \mathsf{F.Delegate})$ *is a delegatable pseudorandom function. Then it holds that* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is a selectively secure HIBE scheme.*

Proof. For the sake of contradiction, we proceed by assuming that there exists a PPT adversary $\mathcal{A}$ such that

$$\Pr[\mathrm{IND}^{\mathrm{HIBE}}_{\mathcal{A}}(1^{\lambda}) = 1] \geq \frac{1}{2} + \epsilon$$

for a non-negligible $\epsilon = \epsilon(\lambda)$, where $\mathrm{IND}^{\mathrm{HIBE}}_{\mathcal{A}}$ is shown in Figure 5. Assume further that q is a polynomial upper bound for the running-time of $\mathcal{A}$, and thus also an upper bound for the number of $\mathcal{A}$'s key queries. Security follows by a sequence of hybrids. In our hybrids, changes are made in how the secret key queries of the adversary $\mathcal{A}$ are answered and how the challenge ciphertext is generated. However, unlike the IBE case, these changes are not intertwined with each other. In particular, we will make changes to the secret keys first and then to the challenge ciphertext. We describe our hybrids next. Our proof consists of a sequence of hybrids $\mathcal{H}_{-2}, \mathcal{H}_{-1}, \mathcal{H}_0, \mathcal{H}_1, \dots \mathcal{H}_{n+2}$.

We describe these below. Since we are in the selective the case the adversary declares the challenge identity ID^* before the public parameters mpk are provided to it. Also, we let V^* be the set $\{\varepsilon, \mathsf{ID}^*[1] \ldots \mathsf{ID}^*[1 \ldots n-1]\}$.

- $\mathcal{H}_{-2}$: This hybrid corresponds to the experiment $\mathrm{IND}^{\mathrm{HIBE}}_{\mathcal{A}}$, as shown in Figure 5.
- $\mathcal{H}_{-1}$: In this hybrid, we replace the pseudorandom function $\mathsf{F}(s, \cdot)$ on ID^* and all prefixes of ID^* with a truly random function $R(\cdot)$ with the same input and output domains. Indistinguishability between hybrids $\mathcal{H}_{-2}$ and $\mathcal{H}_{-1}$ follows based on the selective pseudorandomness of the delegatable pseudorandom function F.
- $\mathcal{H}_0$: In this hybrid, we change how NodeGen and NodeGen′ behave when computed with an input $\mathsf{v} \in V^*$. At a high level, the goal is to change the generating of $\{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v} \in V^*}$ such that the trapdoor values $\mathsf{t}_{\mathsf{v} \in V^*}$ do not need to be used and so that the encryption key $\mathsf{ek}_{\mathsf{ID}^*}$ is sampled independently of everything else. The executions of NodeGen and NodeGen′ for every $\mathsf{v} \notin V^*$ remain unaffected. In particular, at Setup time, we proceed as follows and fix the values $\{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v} \in V^*}$ and $\{\mathsf{dk}_{\mathsf{v}\|0}, \mathsf{dk}_{\mathsf{v}\|1}\}_{\mathsf{v} \in V^*}$.[26]
  (1) For every $\mathsf{v} \in V^*$:
    (a) Generate $(\mathsf{k}_\mathsf{v}, \mathsf{t}_\mathsf{v}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \lambda)$.
    (b) Generate $(\mathsf{ek}_{\mathsf{v}\|0}, \mathsf{dk}_{\mathsf{v}\|0}) \xleftarrow{\$} \mathsf{G}(1^\lambda)$ and $(\mathsf{ek}_{\mathsf{v}\|1}, \mathsf{dk}_{\mathsf{v}\|1}) \xleftarrow{\$} \mathsf{G}(1^\lambda)$.
    (c) Sample $r'_\mathsf{v}, r_\mathsf{v}$.
  (2) Let $S^* := \{\mathsf{ID}^*[1 \ldots i-1] \| (1 - \mathsf{ID}^*[i])\}_{i \in [n]} \cup \{\mathsf{ID}^*\}$. (Note that $S^* \cap V^* = \emptyset$.)
  (3) For all $\mathsf{v} \in S^*$ set $\mathsf{k}_\mathsf{v}, \mathsf{h}_\mathsf{v}$ as first two outputs of $\mathsf{NodeGen}(\mathsf{k}_G, \mathsf{v}, (u, u_0, u_1))$, where $u := \mathsf{F}(s, \mathsf{v})$, $u_0 := \mathsf{F}(s, \mathsf{v}\|0)$ and $u_1 := \mathsf{F}(s, \mathsf{v}\|1)$.
  (4) For each $i \in \{n-1 \ldots 0\}$:
    (a) Set $\mathsf{v} := \mathsf{ID}^*[1 \ldots i]$.
    (b) Generate $\mathsf{h}'_\mathsf{v} := \mathsf{H}(\mathsf{k}_G, \mathsf{k}_{\mathsf{v}\|0} || \mathsf{h}_{\mathsf{v}\|0} || \mathsf{k}_{\mathsf{v}\|1} || \mathsf{h}_{\mathsf{v}\|1} \| \mathsf{ek}_{\mathsf{v}\|0} \| \mathsf{ek}_{\mathsf{v}\|1}; r'_\mathsf{v})$.
    (c) $\mathsf{h}_\mathsf{v} := \mathsf{H}(\mathsf{k}_\mathsf{v}, \mathsf{h}'_\mathsf{v}; r_\mathsf{v})$.
    (d) $\mathsf{lk}_\mathsf{v} := (\mathsf{k}_\mathsf{v}, \mathsf{h}_\mathsf{v}, r_\mathsf{v}, \mathsf{h}'_\mathsf{v}, r'_\mathsf{v}, \mathsf{k}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{k}_{\mathsf{v}\|1}, \mathsf{h}_{\mathsf{v}\|1}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1})$.
  (5) Output $\{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v} \in V^*}$ and $\{\mathsf{dk}_{\mathsf{v}\|0}, \mathsf{dk}_{\mathsf{v}\|1}\}_{\mathsf{v} \in V^*}$.
  Statistical indistinguishability of hybrids $\mathcal{H}_{-1}$ and $\mathcal{H}_0$ follows from the trapdoor collision and uniformity properties of the chameleon encryption scheme. Note that in this hybrid the trapdoor t_v for any node $\mathsf{v} \in V^*$ is no longer being used.
- $\mathcal{H}_\tau$ for $\tau \in \{1 \ldots n\}$: This hybrid is identical to $\mathcal{H}_0$, except that we change how the challenge ciphertext is generated. Recall that the challenge ciphertext consists of a sequence of $2n+1$ garbled circuits. In hybrid $\mathcal{H}_\tau$, we generate the first 2τ of these garbled circuits (namely, $\tilde{P}^1, \tilde{Q}^1 \ldots \tilde{P}^\tau, \tilde{Q}^\tau$) using the simulator provided by the garbled circuit construction. The outputs hard-coded in the simulated circuits are set to be consistent with the output that would have resulted from the execution of honestly generated garbled circuits using keys obtained from invocations of NodeGen. More formally, for the challenge identity ID^* the challenge ciphertext is generated as follows (modifications with respect to honest ciphertext generation have been highlighted in red):
  (1) Compute $\tilde{T}$ as:

$$(\tilde{T}, \overline{\mathsf{tlab}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{T}[\mathsf{m}]).$$

  (2) For $i = n, \ldots, \tau + 1$ generate

[26] Note that the adversary never makes a KeyGen query for an identity ID that is a prefix of ID^*. Therefore, we have that dk_v for $\mathsf{v} \in V^* \cup \{\mathsf{ID}^*\}$ will not be provided to $\mathcal{A}$.

(a) If $i = n$, then

$$(\tilde{Q}^n, \overline{\mathsf{qlab}}^n) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{Q}_{last}[\mathsf{ID}[n], \mathsf{k}_G, \overline{\mathsf{tlab}}]),$$

else

$$(\tilde{Q}^i, \overline{\mathsf{qlab}}^i) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{Q}[\mathsf{ID}[i], \mathsf{k}_G, \overline{\mathsf{plab}}^{i+1}]).$$

(b) $(\tilde{P}^i, \overline{\mathsf{plab}}^i) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{P}[\overline{\mathsf{qlab}}^i])$.

(3) For $i = \tau, \ldots, 1$:

(a) Set $\mathsf{v} = \mathsf{ID}^*[1 \ldots i-1], x_\mathsf{v} := \mathsf{k}_\mathsf{v} \| \mathsf{h}_\mathsf{v}, y_\mathsf{v} := \mathsf{h}'_\mathsf{v}$, and if $i < n$ then $z_\mathsf{v} := \mathsf{k}_{\mathsf{v}\|\mathsf{ID}^*[i]} \| \mathsf{h}_{\mathsf{v}\|\mathsf{ID}^*[i]}$ else $z_\mathsf{v} := \mathsf{ek}_{\mathsf{ID}^*}$.

(b) If $i = n$ then

$$(\tilde{Q}^n, \{\mathsf{qlab}^n_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_\mathsf{v}, j + \mathsf{ID}^*[n]\cdot\lambda + 2\ell, b), \mathsf{tlab}_{j,z_{\mathsf{v},j}})\}_{j\in[\lambda], b\in\{0,1\}})$$

else

$$(\tilde{Q}^i, \{\mathsf{qlab}^i_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_\mathsf{v}, j + \mathsf{ID}^*[i]\cdot\ell, b), \mathsf{plab}^{i+1}_{j,z_{\mathsf{v},j}})\}_{j\in[\ell], b\in\{0,1\}}).$$

(c) $\overline{\mathsf{qlab}}^i := \{\mathsf{qlab}^i_{j,y_{\mathsf{v},j}}, \mathsf{qlab}^i_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}$.

(d) $(\tilde{P}^i, \{\mathsf{plab}^i_{j,x_{\mathsf{v},j}}\}_{j\in[\ell]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k}_\mathsf{v}, (\mathsf{h}_\mathsf{v}, j, b), \mathsf{qlab}^i_{j,y_{\mathsf{v},j}})\}_{j\in[\lambda], b\in\{0,1\}})$.

(e) $\overline{\mathsf{plab}}^i := \{\mathsf{plab}^i_{j,x_{\mathsf{v},j}}, \mathsf{plab}^i_{j,x_{\mathsf{v},j}}\}_{j\in[\ell]}$.

(4) Set $x_\varepsilon := \mathsf{k}_\varepsilon \| \mathsf{h}_\varepsilon$.

(5) Output $\mathsf{ct} := (\{\mathsf{plab}^1_{j,x_{\varepsilon,j}}\}_{j\in[\lambda]}, \{\tilde{P}^i, \tilde{Q}^i\}_{i\in[n]}, \tilde{T})$ where $x_{\varepsilon,j}$ is the jth bit of x_ε.

The computational indistinguishability between hybrids $\mathcal{H}_{\tau-1}$ and $\mathcal{H}_\tau$ is based on Lemma 7.2, which is proved in Section 7.3.

Lemma 7.2. *For each $\tau \in \{1 \ldots n\}$ it is the case that $\mathcal{H}_{\tau-1} \overset{c}{\approx} \mathcal{H}_\tau$.*

- $\mathcal{H}_{n+1}$: This hybrid is same as hybrid $\mathcal{H}_n$ except that we generate the garbled circuit $\tilde{T}$ to use the garbling simulator. More specifically, instead of generating $\tilde{T}$ as

$$(\tilde{T}, \overline{\mathsf{tlab}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{T}[\mathsf{m}]),$$

we set $y = \mathsf{ek}_{\mathsf{ID}^*}$ and generate garbled circuit as,

$$(\tilde{T}, \{\mathsf{lab}_{j,y_j}\}_{j\in[\lambda]}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{E}(y, \mathsf{m}))$$

and set $\overline{\mathsf{lab}} := \{\mathsf{lab}_{j,y_j}, \mathsf{lab}_{j,y_j}\}_{j\in[\lambda]}$.

Computational indistinguishability between hybrids $\mathcal{H}_n$ and $\mathcal{H}_{n+1}$ follows directly from the security of the garbled circuits.

- $\mathcal{H}_{n+2}$: This hybrid is same as $\mathcal{H}_{n+1}$ except that we change the ciphertext $\mathsf{E}(\mathsf{ek}_{\mathsf{ID}^*}, \mathsf{m})$ hard-wired in the simulated garbling of the circuit T to be $\mathsf{E}(\mathsf{ek}_{\mathsf{ID}^*}, 0)$.

  Note that the adversary $\mathcal{A}$ never queries the key-generation oracle for $\mathsf{sk}_{\mathsf{ID}^*}$ or any $\mathsf{sk}_{\mathsf{ID}'}$ such that ID' is a prefix of ID^*. Therefore, it is never provided the value $\mathsf{dk}_{\mathsf{ID}^*}$. Therefore, we can use an adversary distinguishing between $\mathcal{H}_{n+1}$ and $\mathcal{H}_{n+2}$ to construct an attacker against the semantic security of the public-key encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$. This allows us to conclude that $\mathcal{H}_{n+1} \overset{c}{\approx} \mathcal{H}_{n+2}$.

  Finally, note that the hybrid $\mathcal{H}_{n+2}$ is information theoretically independent of the plaintext message m.

7.3 Proof of Lemma 7.2

The proof proceeds by a sequence of sub-hybrids $\mathcal{H}_{\tau,0}$ to $\mathcal{H}_{\tau,4}$, where $\mathcal{H}_{\tau,0}$ is same as $\mathcal{H}_{\tau-1}$ and $\mathcal{H}_{\tau,4}$ is same as $\mathcal{H}_{\tau}$.

- $\mathcal{H}_{\tau,0}$: This hybrid is same as $\mathcal{H}_{\tau-1}$.
- $\mathcal{H}_{\tau,1}$: In this hybrid, we change how the garbled circuit $\tilde{P}^{\tau}$ is generated. Let $\mathsf{v} = \mathsf{ID}^*[1 \ldots \tau-1]$, $\mathsf{lk}_{\mathsf{v}} = (\mathsf{k}_{\mathsf{v}}, \mathsf{h}_{\mathsf{v}}, r_{\mathsf{v}}, \mathsf{h}'_{\mathsf{v}}, r'_{\mathsf{v}}, \mathsf{k}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{k}_{\mathsf{v}\|1}, \mathsf{h}_{\mathsf{v}\|1}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1})$ and define $x_{\mathsf{v}} := \mathsf{k}_{\mathsf{v}}\|\mathsf{h}_{\mathsf{v}}$. The change we make is the following: We generate

  $$(\tilde{P}^{\tau}, \overline{\mathsf{plab}}^{\tau}) \xleftarrow{\$} \mathsf{GCircuit}(1^{\lambda}, \mathsf{P}[\overline{\mathsf{qlab}}^{\tau}])$$

  now as

  $$(\tilde{P}^{\tau}, \{\mathsf{plab}^{\tau}_{j,x_{\mathsf{v},j}}\}_{j\in[\ell]}) \xleftarrow{\$} \mathsf{Sim}(1^{\lambda}, \{\mathsf{Enc}(\mathsf{k}_{\mathsf{v}}, (\mathsf{h}_{\mathsf{v}}, j, b), \mathsf{qlab}^{\tau}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}),$$

  where $x_{\mathsf{v},j}$ is the jth bit of x_{v}. Next, we set $\overline{\mathsf{plab}}^{\tau} := \{\mathsf{plab}^{\tau}_{j,x_{\mathsf{v},j}}, \mathsf{plab}^{\tau}_{j,x_{\mathsf{v},j}}\}_{j\in[\ell]}$.
  Computational indistinguishability of hybrids $\mathcal{H}_{\tau,0}$ and $\mathcal{H}_{\tau,1}$ follows by the security of the garbling scheme GCircuit and the fact that $\{\mathsf{Enc}(\mathsf{k}_{\mathsf{v}}, (\mathsf{h}_{\mathsf{v}}, j, b), \mathsf{qlab}^{\tau}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}$ is exactly the output of the circuit $P[\overline{\mathsf{qlab}}^{\tau}]$ on input x_{v}.
- $\mathcal{H}_{\tau,2}$: This hybrid is identical to $\mathcal{H}_{\tau,2}$, except that for $\mathsf{v} = \mathsf{ID}^*[1 \ldots \tau-1]$, we change

  $$(\tilde{P}^{\tau}, \{\mathsf{plab}^{\tau}_{j,x_{\mathsf{v},j}}\}_{j\in[\ell]}) := \mathsf{Sim}(1^{\lambda}, \{\mathsf{Enc}(\mathsf{k}_{\mathsf{v}}, (\mathsf{h}_{\mathsf{v}}, j, b), \mathsf{qlab}^{\tau}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}})$$

  to

  $$(\tilde{P}^{\tau}, \{\mathsf{plab}^{\tau}_{j,x_{\mathsf{v},j}}\}_{j\in[\ell]}) := \mathsf{Sim}(1^{\lambda}, \{\mathsf{Enc}(\mathsf{k}_{\mathsf{v}}, (\mathsf{h}_{\mathsf{v}}, j, b), \mathsf{qlab}^{\tau}_{j,y_{\mathsf{v},j}})\}_{j\in[\lambda], b\in\{0,1\}})$$

  where $y_{\mathsf{v}} := \mathsf{h}'_{\mathsf{v}}$.
  Notice that node v is generated so the trapdoor value t_{v} is not used in the execution of the experiment. Therefore, computational indistinguishability of hybrids $\mathcal{H}_{\tau,1}$ and $\mathcal{H}_{\tau,2}$ follows by λ invocations (one invocation for each bit of the λ labels) of the security of the chameleon encryption scheme. The reduction is analogous to the reduction proving indistinguishability of hybrids $\mathcal{H}_{\tau,2}$ and $\mathcal{H}_{\tau,3}$ in the proof of Lemma 6.2.
  Remark. We note that the ciphertexts hardwired inside the garbled circuit only provide the labels $\{\mathsf{qlab}^{\tau}_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}$ (in an information theoretical sense).
- $\mathcal{H}_{\tau,3}$ This hybrid is identical to $\mathcal{H}_{\tau,2}$, except that for $\mathsf{v} = \mathsf{ID}^*[1 \ldots \tau-1]$, we change how $\tilde{Q}^{\tau}$ is generated. If $\tau = n$, then

  $$(\tilde{Q}^{n}, \overline{\mathsf{qlab}}^{n}) \xleftarrow{\$} \mathsf{GCircuit}(1^{\lambda}, \mathsf{Q}_{last}[\mathsf{ID}^*[n], \mathsf{k}_G, \overline{\mathsf{tlab}}]),$$

  is changed to

  $$(\tilde{Q}^{n}, \{\mathsf{qlab}^{n}_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^{\lambda}, \{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_{\mathsf{v}}, j + \mathsf{ID}^*[n]\cdot\lambda + 2\ell, b), \mathsf{tlab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}),$$

  and $\overline{\mathsf{qlab}}^{n} := \{\mathsf{qlab}^{n}_{j,y_{\mathsf{v},j}}, \mathsf{qlab}^{n}_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}$ where $y_{\mathsf{v}} := \mathsf{h}'_{\mathsf{v}}$. Otherwise, if $\tau \neq n$, then

  $$(\tilde{Q}^{\tau}, \overline{\mathsf{qlab}}^{\tau}) \xleftarrow{\$} \mathsf{GCircuit}(1^{\lambda}, \mathsf{Q}[\mathsf{ID}^*[\tau], \mathsf{k}_G, \overline{\mathsf{plab}}^{\tau+1}])$$

  is changed to

  $$(\tilde{Q}^{\tau}, \{\mathsf{qlab}^{\tau}_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^{\lambda}, \{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_{\mathsf{v}}, j + \mathsf{ID}^*[\tau]\cdot\ell, b), \mathsf{plab}^{\tau+1}_{j,b})\}_{j\in[\ell], b\in\{0,1\}}),$$

  and $\overline{\mathsf{qlab}}^{\tau} := \{\mathsf{qlab}^{\tau}_{j,y_{\mathsf{v},j}}, \mathsf{qlab}^{\tau}_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}$ where $y_{\mathsf{v}} := \mathsf{h}'_{\mathsf{v}}$.

Computational indistinguishability between hybrids $\mathcal{H}_{\tau,2}$ and $\mathcal{H}_{\tau,3}$ follows by the security of the garbling scheme and the fact that is the output of the circuit $\mathsf{Q}_{last}[\mathsf{ID}^*[n], \mathsf{k}_G, \overline{\mathsf{tlab}}]$ is $\{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_{\mathsf{v}}, j + \mathsf{ID}^*[n] \cdot \lambda + 2\ell, b), \mathsf{tlab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}$ and the output of the circuit $\mathsf{Q}[\mathsf{ID}^*[\tau], \mathsf{k}_G, \overline{\mathsf{plab}}^{\tau+1}]$ is $\{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_{\mathsf{v}}, j + \mathsf{ID}^*[\tau] \cdot \ell, b), \mathsf{plab}^{\tau+1}_{j,b})\}_{j\in[\ell], b\in\{0,1\}}$.

- $\mathcal{H}_{\tau,4}$: This hybrid is identical to $\mathcal{H}_{\tau,4}$, except that we change generation of $\tilde{Q}^\tau$. Specifically, in the case $\tau = n$ then we change the generation process of $\tilde{Q}^n$ from

  $$(\tilde{Q}^n, \{\mathsf{qlab}^n_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_{\mathsf{v}}, j + \mathsf{ID}^*[n] \cdot \lambda + 2\ell, b), \mathsf{tlab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}})$$

  to

  $$(\tilde{Q}^n, \{\mathsf{qlab}^n_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_{\mathsf{v}}, j + \mathsf{ID}^*[n] \cdot \lambda + 2\ell, b), \mathsf{tlab}_{j,z_{\mathsf{v},j}})\}_{j\in[\lambda], b\in\{0,1\}}),$$

  where $z_{\mathsf{v}} := \mathsf{ek}_{\mathsf{ID}^*}$. However, when $\tau \neq n$ then it is changed from

  $$(\tilde{Q}^\tau, \{\mathsf{qlab}^\tau_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_{\mathsf{v}}, j + \mathsf{ID}^*[\tau] \cdot \ell, b), \mathsf{plab}^{\tau+1}_{j,b})\}_{j\in[\ell], b\in\{0,1\}})$$

  to

  $$(\tilde{Q}^\tau, \{\mathsf{qlab}^\tau_{j,y_{\mathsf{v},j}}\}_{j\in[\lambda]}) := \mathsf{Sim}(1^\lambda, \{\mathsf{Enc}(\mathsf{k}_G, (\mathsf{h}'_{\mathsf{v}}, j + \mathsf{ID}^*[\tau] \cdot \ell, b), \mathsf{plab}^{\tau+1}_{j,z_{\mathsf{v},j}})\}_{j\in[\ell], b\in\{0,1\}}),$$

  where $z_{\mathsf{v}} := \mathsf{h}_{\mathsf{v}\|\mathsf{ID}^*[\tau]}\|\mathsf{k}_{\mathsf{v}\|\mathsf{ID}^*[\tau]}$.
  Notice that, since the trapdoor for k_G is unavailable (never generated or used), computational indistinguishability of hybrids $\mathcal{H}_{\tau,3}$ and $\mathcal{H}_{\tau,4}$ follows by λ^2 invocations (one invocation per bit of the λ labels) if $\tau = n$ and by $\ell\lambda$ invocations (one invocation per bit of the ℓ labels) of the security of the chameleon encryption scheme. And the reduction to the security of the chameleon encryption scheme is analogous to the reduction described for indistinguishability between hybrids $\mathcal{H}_{\tau,1}$ and $\mathcal{H}_{\tau,2}$.
  Observe that the hybrid $\mathcal{H}_{\tau,4}$ is the same as hybrid $\mathcal{H}_\tau$.

REFERENCES

Leonard Adleman. 1979. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *20th Symposium on Foundations of Computer Science (SFCS'79)*. IEEE, 55–60.

Shweta Agrawal, Dan Boneh, and Xavier Boyen. 2010a. Efficient lattice (H)IBE in the standard model. In *Advances in Cryptology – EUROCRYPT'10 (Lecture Notes in Computer Science)*, Henri Gilbert (Ed.), Vol. 6110. Springer, Germany, 553–572. DOI: https://doi.org/10.1007/978-3-642-13190-5_28

Shweta Agrawal, Dan Boneh, and Xavier Boyen. 2010b. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *Advances in Cryptology – CRYPTO'10 (Lecture Notes in Computer Science)*, Tal Rabin (Ed.), Vol. 6223. Springer, Germany, 98–115. DOI: https://doi.org/10.1007/978-3-642-14623-7_6

Shweta Agrawal and Xavier Boyen. 2009. Identity-based encryption from lattices in the standard model. *Manuscript*. Retrieved from http://www.cs.stanford.edu/ xb/ab09/.

William Aiello, Yuval Ishai, and Omer Reingold. 2001. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology – EUROCRYPT'01 (Lecture Notes in Computer Science)*, Birgit Pfitzmann (Ed.), Vol. 2045. Springer, Germany, 119–135. DOI: https://doi.org/10.1007/3-540-44987-6_8

Adi Akavia, Shafi Goldwasser, and Shmuel Safra. 2003. Proving hard-core predicates using list decoding. In *44th Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 146–159. DOI: https://doi.org/10.1109/SFCS.2003.1238189

Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. 2013. Differing-Inputs Obfuscation and Applications. *Cryptology ePrint Archive, Report 2013/689*. Retrieved from http://eprint.iacr.org/2013/689.

Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. 2014. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In *Advances in Cryptology – EUROCRYPT'14 (Lecture Notes in Computer Science)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.), Vol. 8441. Springer, Germany, 1–16. DOI: https://doi.org/10.1007/978-3-642-55220-5_1

Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of garbled circuits. In *ACM CCS'12: 19th Conference on Computer and Communications Security*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM Press, 784–796. DOI: https://doi.org/10.1145/2382196.2382279

Mihir Bellare and Silvio Micali. 1990. Non-interactive oblivious transfer and applications. In *Advances in Cryptology – CRYPTO'89 (Lecture Notes in Computer Science)*, Gilles Brassard (Ed.), Vol. 435. Springer, Germany, 547–557. DOI: https://doi.org/10.1007/0-387-34805-0_48

Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS'93: 1st Conference on Computer and Communications Security*, Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby (Eds.). ACM Press, 62–73. DOI: https://doi.org/10.1145/168588.168596

Eli Biham, Dan Boneh, and Omer Reingold. 1997. Generalized Diffie-Hellman Modulo a Composite is not Weaker than Factoring. *Cryptology ePrint Archive, Report 1997/014*. Retrieved from http://eprint.iacr.org/1997/014.

Dan Boneh and Xavier Boyen. 2004a. Efficient selective-ID secure identity based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT'04 (Lecture Notes in Computer Science)*, Christian Cachin and Jan Camenisch (Eds.), Vol. 3027. Springer, Germany, 223–238. DOI: https://doi.org/10.1007/978-3-540-24676-3_14

Dan Boneh and Xavier Boyen. 2004b. Secure identity based encryption without random oracles. In *Advances in Cryptology – CRYPTO'04 (Lecture Notes in Computer Science)*, Matthew Franklin (Ed.), Vol. 3152. Springer, Germany, 443–459. DOI: https://doi.org/10.1007/978-3-540-28628-8_27

Dan Boneh, Xavier Boyen, and Eu-Jin Goh. 2005. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology – EUROCRYPT'05 (Lecture Notes in Computer Science)*, Ronald Cramer (Ed.), Vol. 3494. Springer, Germany, 440–456. DOI: https://doi.org/10.1007/11426639_26

Dan Boneh and Matthew K. Franklin. 2001. Identity-based encryption from the Weil pairing. In *Advances in Cryptology – CRYPTO'01 (Lecture Notes in Computer Science)*, Joe Kilian (Ed.), Vol. 2139. Springer, Germany, 213–229. DOI: https://doi.org/10.1007/3-540-44647-8_13

Dan Boneh, Craig Gentry, and Michael Hamburg. 2007. Space-efficient identity based encryption without pairings. In *48th Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 647–657. DOI: https://doi.org/10.1109/FOCS.2007.64

Dan Boneh, Periklis A. Papakonstantinou, Charles Rackoff, Yevgeniy Vahlis, and Brent Waters. 2008. On the impossibility of basing identity based encryption on trapdoor permutations. In *49th Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 283–292. DOI: https://doi.org/10.1109/FOCS.2008.67

Elette Boyle, Kai-Min Chung, and Rafael Pass. 2014a. On extractability obfuscation. In *TCC'14: 11th Theory of Cryptography Conference (Lecture Notes in Computer Science)*, Yehuda Lindell (Ed.), Vol. 8349. Springer, Germany, 52–73. DOI: https://doi.org/10.1007/978-3-642-54242-8_3

Elette Boyle, Shafi Goldwasser, and Ioana Ivan. 2014b. Functional signatures and pseudorandom functions. In *PKC'14: 17th International Conference on Theory and Practice of Public Key Cryptography (Lecture Notes in Computer Science)*, Hugo Krawczyk (Ed.), Vol. 8383. Springer, Germany, 501–519. DOI: https://doi.org/10.1007/978-3-642-54631-0_29

Gilles Brassard, David Chaum, and Claude Crépeau. 1988. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.* 37, 2 (Oct. 1988), 156–189. DOI: https://doi.org/10.1016/0022-0000(88)90005-0

Ran Canetti, Oded Goldreich, and Shai Halevi. 1998. The random oracle methodology, revisited (preliminary version). In *30th ACM Symposium on Theory of Computing*. ACM Press, 209–218. DOI: https://doi.org/10.1145/276698.276741

Ran Canetti, Shai Halevi, and Jonathan Katz. 2003. A forward-secure public-key encryption scheme. In *Advances in Cryptology – EUROCRYPT'03 (Lecture Notes in Computer Science)*, Eli Biham (Ed.), Vol. 2656. Springer, Germany, 255–271. DOI: https://doi.org/10.1007/3-540-39200-9_16

Ran Canetti, Shai Halevi, and Jonathan Katz. 2004. Chosen-ciphertext security from identity-based encryption. In *Advances in Cryptology – EUROCRYPT'04 (Lecture Notes in Computer Science)*, Christian Cachin and Jan Camenisch (Eds.), Vol. 3027. Springer, Germany, 207–222. DOI: https://doi.org/10.1007/978-3-540-24676-3_13

David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. 2010. Bonsai trees, or how to delegate a lattice basis. In *Advances in Cryptology – EUROCRYPT'10 (Lecture Notes in Computer Science)*, Henri Gilbert (Ed.), Vol. 6110. Springer, Germany, 523–552. DOI: https://doi.org/10.1007/978-3-642-13190-5_27

Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. 2017. Laconic oblivious transfer and its applications. In *Advances in Cryptology – CRYPTO'17, Part II (Lecture Notes in Computer Science)*, Jonathan Katz and Hovav Shacham (Eds.), Vol. 10402. Springer, Germany, 33–65. DOI: https://doi.org/10.1007/978-3-319-63715-0_2

Clifford Cocks. 2001. An identity based encryption scheme based on quadratic residues. In *8th IMA International Conference on Cryptography and Coding (Lecture Notes in Computer Science)*, Bahram Honary (Ed.), Vol. 2260. Springer, Germany, 360–363.

Henri Cohen. 2013. *A Course in Computational Algebraic Number Theory*. Vol. 138. Springer Science & Business Media.

An Commeine and Igor Semaev. 2006. An algorithm to solve the discrete logarithm problem with the number field sieve. In *PKC'06: 9th International Conference on Theory and Practice of Public Key Cryptography (Lecture Notes in Computer Science)*, Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin (Eds.), Vol. 3958. Springer, Germany, 174–190. DOI: https://doi.org/10.1007/11745853_12

Whitfield Diffie and Martin E. Hellman. 1976. New directions in cryptography. *IEEE Trans. Inf. Theor.* 22, 6 (1976), 644–654.

Nico Döttling and Sanjam Garg. 2017. From selective IBE to full IBE and selective HIBE. In *TCC'17: 15th Theory of Cryptography Conference, Part I (Lecture Notes in Computer Science)*, Yael Kalai and Leonid Reyzin (Eds.), Vol. 10677. Springer, Germany, 372–408. DOI: https://doi.org/10.1007/978-3-319-70500-2_13

Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. 2018. New constructions of identity-based and key-dependent message secure encryption schemes. In *PKC'18: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I (Lecture Notes in Computer Science)*, Michel Abdalla and Ricardo Dahab (Eds.), Vol. 10769. Springer, Germany, 3–31. DOI: https://doi.org/10.1007/978-3-319-76578-5_1

Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. 2013. Witness encryption and its applications. In *45th ACM Symposium on Theory of Computing*, Dan Boneh, Tim Roughgarden, and Joan Feigenbaum (Eds.). ACM Press, 467–476. DOI: https://doi.org/10.1145/2488608.2488667

Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. 2018. Registration-based encryption: Removing private-key generator from IBE. In *TCC'18: 16th Theory of Cryptography Conference, Part I (Lecture Notes in Computer Science)*, Amos Beimel and Stefan Dziembowski (Eds.), Vol. 11239. Springer, Germany, 689–718. DOI: https://doi.org/10.1007/978-3-030-03807-6_25

Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. 2019. Registration-based encryption from standard assumptions. In *PKC'19: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part II (Lecture Notes in Computer Science)*, Dongdai Lin and Kazue Sako (Eds.), Vol. 11443. Springer, Germany, 63–93. DOI: https://doi.org/10.1007/978-3-030-17259-6_3

Sanjam Garg, Steve Lu, and Rafail Ostrovsky. 2015a. Black-box garbled RAM. In *56th Symposium on Foundations of Computer Science*, Venkatesan Guruswami (Ed.). IEEE Computer Society Press, 210–229. DOI: https://doi.org/10.1109/FOCS.2015.22

Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. 2015b. Garbled RAM from one-way functions. In *47th ACM Symposium on Theory of Computing*, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM Press, 449–458. DOI: https://doi.org/10.1145/2746539.2746593

Craig Gentry and Shai Halevi. 2009. Hierarchical identity based encryption with polynomially many levels. In *TCC'09: 6th Theory of Cryptography Conference (Lecture Notes in Computer Science)*, Omer Reingold (Ed.), Vol. 5444. Springer, Germany, 437–456. DOI: https://doi.org/10.1007/978-3-642-00457-5_26

Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. 2014. Garbled RAM revisited. In *Advances in Cryptology – EUROCRYPT'14 (Lecture Notes in Computer Science)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.), Vol. 8441. Springer, Germany, 405–422. DOI: https://doi.org/10.1007/978-3-642-55220-5_23

Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. 2008. Trapdoors for hard lattices and new cryptographic constructions. In *40th ACM Symposium on Theory of Computing*, Richard E. Ladner and Cynthia Dwork (Eds.). ACM Press, 197–206. DOI: https://doi.org/10.1145/1374376.1374407

Craig Gentry and Alice Silverberg. 2002. Hierarchical ID-based cryptography. In *Advances in Cryptology – ASIACRYPT'02 (Lecture Notes in Computer Science)*, Yuliang Zheng (Ed.), Vol. 2501. Springer, Germany, 548–566. DOI: https://doi.org/10.1007/3-540-36178-2_34

Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1984. How to construct random functions (extended abstract). In *25th Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 464–479. DOI: https://doi.org/10.1109/SFCS.1984.715949

Oded Goldreich and Leonid A. Levin. 1989. A hard-core predicate for all one-way functions. In *21st ACM Symposium on Theory of Computing*. ACM Press, 25–32. DOI: https://doi.org/10.1145/73007.73010

Shafi Goldwasser and Silvio Micali. 1982. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM Symposium on Theory of Computing*. ACM Press, 365–377. DOI: https://doi.org/10.1145/800070.802212

Daniel M. Gordon. 1993. Discrete logarithms in GF(P) using the number field sieve. *SIAM J. Disc. Math.* 6, 1 (1993), 124–138.

Helmut Hasse. 1936. Zur Theorie der abstrakten elliptischen Funktionenkörper III. Die Struktur des Meromorphismenrings. Die Riemannsche Vermutung. *J. reine und angewandte Math.* 175 (1936), 193–208.

Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. 1999. A pseudorandom generator from any one-way function. *SIAM J. Comput.* 28, 4 (1999), 1364–1396.

Martin E. Hellman and Justin M. Reyneri. 1983. Fast computation of discrete logarithms in GF (q). In *Advances in Cryptology*. Springer, 3–13.

Dennis Hofheinz and Eike Kiltz. 2007. Secure hybrid encryption from weakened key encapsulation. In *Advances in Cryptology – CRYPTO'07 (Lecture Notes in Computer Science)*, Alfred Menezes (Ed.), Vol. 4622. Springer, Germany, 553–571. DOI: https://doi.org/10.1007/978-3-540-74143-5_31

Dennis Hofheinz and Eike Kiltz. 2009. The group of signed quadratic residues and applications. In *Advances in Cryptology – CRYPTO'09 (Lecture Notes in Computer Science)*, Shai Halevi (Ed.), Vol. 5677. Springer, Germany, 637–653. DOI: https://doi.org/10.1007/978-3-642-03356-8_37

Jeremy Horwitz and Ben Lynn. 2002. Toward hierarchical identity-based encryption. In *Advances in Cryptology – EUROCRYPT'02 (Lecture Notes in Computer Science)*, Lars R. Knudsen (Ed.), Vol. 2332. Springer, Germany, 466–481. DOI: https://doi.org/10.1007/3-540-46035-7_31

Russell Impagliazzo, Leonid A. Levin, and Michael Luby. 1989. Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM Symposium on Theory of Computing*. ACM Press, 12–24. DOI: https://doi.org/10.1145/73007.73009

Antoine Joux. 2000. A one round protocol for tripartite Diffie-Hellman. In *ANTS (Lecture Notes in Computer Science)*, Vol. 1838. Springer, 385–394.

Antoine Joux and Reynald Lercier. 2003. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method. *Math. Comput.* 72, 242 (2003), 953–967.

Antoine Joux, Reynald Lercier, Nigel Smart, and Frederik Vercauteren. 2006. The number field sieve in the medium prime case. In *Advances in Cryptology – CRYPTO'06 (Lecture Notes in Computer Science)*, Cynthia Dwork (Ed.), Vol. 4117. Springer, Germany, 326–344. DOI: https://doi.org/10.1007/11818175_19

Neal Koblitz. 1987. Elliptic curve cryptosystems. *Math. Comp.* 48, 177 (1987), 203–209. Retrieved from http://www.jstor.org/stable/2007884.

Hugo Krawczyk and Tal Rabin. 1998. Chameleon Hashing and Signatures. *Cryptology ePrint Archive, Report 1998/010*. Retrieved from http://eprint.iacr.org/1998/010.

Eyal Kushilevitz and Yishay Mansour. 1991. Learning decision trees using the Fourier sprectrum (extended abstract). In *23rd ACM Symposium on Theory of Computing*. ACM Press, 455–464. DOI: https://doi.org/10.1145/103418.103466

Allison B. Lewko and Brent Waters. 2010. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In *TCC'10: 7th Theory of Cryptography Conference (Lecture Notes in Computer Science)*, Daniele Micciancio (Ed.), Vol. 5978. Springer, Germany, 455–479. DOI: https://doi.org/10.1007/978-3-642-11799-2_27

Yehuda Lindell and Benny Pinkas. 2009. A proof of security of Yao's protocol for two-party computation. *J. Cryptol.* 22, 2 (Apr. 2009), 161–188. DOI: https://doi.org/10.1007/s00145-008-9036-8

Steve Lu and Rafail Ostrovsky. 2013. How to garble RAM programs. In *Advances in Cryptology – EUROCRYPT'13 (Lecture Notes in Computer Science)*, Thomas Johansson and Phong Q. Nguyen (Eds.), Vol. 7881. Springer, Germany, 719–734. DOI: https://doi.org/10.1007/978-3-642-38348-9_42

Kevin S. McCurley. 1988. A key distribution system equivalent to factoring. *J. Cryptol.* 1, 2 (June 1988), 95–105. DOI: https://doi.org/10.1007/BF02351718

Victor S. Miller. 1986. Use of elliptic curves in cryptography. In *Advances in Cryptology – CRYPTO'85 (Lecture Notes in Computer Science)*, Hugh C. Williams (Ed.), Vol. 218. Springer, Germany, 417–426. DOI: https://doi.org/10.1007/3-540-39799-X_31

Moni Naor and Moti Yung. 1989. Universal one-way hash functions and their cryptographic applications. In *21st ACM Symposium on Theory of Computing*. ACM Press, 33–43. DOI: https://doi.org/10.1145/73007.73011

Harald Niederreiter and Chaoping Xing. 2009. *Algebraic Geometry in Coding Theory and Cryptography*. Princeton University Press.

Tatsuaki Okamoto and Katsuyuki Takashima. 2010. Fully secure functional encryption with general relations from the decisional linear assumption. In *Advances in Cryptology – CRYPTO'10 (Lecture Notes in Computer Science)*, Tal Rabin (Ed.), Vol. 6223. Springer, Germany, 191–208. DOI: https://doi.org/10.1007/978-3-642-14623-7_11

Periklis A. Papakonstantinou, Charles W. Rackoff, and Yevgeniy Vahlis. 2012. How powerful are the DDH hard groups?*Cryptology ePrint Archive, Report 2012/653*. Retrieved from http://eprint.iacr.org/2012/653.

Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology – CRYPTO'08 (Lecture Notes in Computer Science)*, David Wagner (Ed.), Vol. 5157. Springer, Germany, 554–571. DOI: https://doi.org/10.1007/978-3-540-85174-5_31

Stephen Carl Pohlig. 1977. *Algebraic and Combinatoric Aspects of Cryptography*. Number 6602. Stanford University.

John M. Pollard. 1978. Monte Carlo methods for index computation (mod p). *Math. Comput.* 32, 143 (1978), 918–924.

Michael O. Rabin. 2005. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive* 2005 (2005), 187.

Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *37th ACM Symposium on Theory of Computing*, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, 84–93. DOI: https://doi.org/10.1145/1060590.1060603

Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. 1978. A method for obtaining digital signature and public-key cryptosystems. *Commun. Assoc. Comput. Mach.* 21, 2 (1978), 120–126.

Adi Shamir. 1984. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology – CRYPTO'84 (Lecture Notes in Computer Science)*, G. R. Blakley and David Chaum (Eds.), Vol. 196. Springer, Germany, 47–53.

Daniel Shanks. 1972. The infrastructure of a real quadratic field and its applications. In *Proceedings of the Number Theory Conference*. 217–224.

Elaine Shi and Brent Waters. 2008. Delegating capabilities in predicate encryption systems. In *ICALP '08: 35th International Colloquium on Automata, Languages and Programming, Part II (Lecture Notes in Computer Science)*, Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz (Eds.), Vol. 5126. Springer, Germany, 560–578. DOI: https://doi.org/10.1007/978-3-540-70583-3_46

Z. Shmuely. 1985. Composite Diffie-Hellman public-key generating systems are hard to break. *Technical Report No. 356, Computer Science Department, Technion, Israel.*

Peter W. Shor. 1994. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 124–134. DOI: https://doi.org/10.1109/SFCS.1994.365700

Victor Shoup. 1997. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology – EUROCRYPT'97 (Lecture Notes in Computer Science)*, Walter Fumy (Ed.), Vol. 1233. Springer, Germany, 256–266. DOI: https://doi.org/10.1007/3-540-69053-0_18

Brent Waters. 2009. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *Advances in Cryptology – CRYPTO'09 (Lecture Notes in Computer Science)*, Shai Halevi (Ed.), Vol. 5677. Springer, Germany, 619–636. DOI: https://doi.org/10.1007/978-3-642-03356-8_36

Brent R. Waters. 2005. Efficient identity-based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT'05 (Lecture Notes in Computer Science)*, Ronald Cramer (Ed.), Vol. 3494. Springer, Germany, 114–127. DOI: https://doi.org/10.1007/11426639_7

Andrew Chi-Chih Yao. 1982. Protocols for secure computations (extended abstract). In *23rd Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 160–164. DOI: https://doi.org/10.1109/SFCS.1982.38

Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets (extended abstract). In *27th Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 162–167. DOI: https://doi.org/10.1109/SFCS.1986.25

Received September 2017; revised October 2019; accepted August 2020