Identity-Based Encryption from the Diffie-Hellman Assumption*

Nico Döttling Sanjam Garg

University of California, Berkeley

Abstract

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.

1 Introduction

Soon after the invention of public-key encryption [DH76,RSA78], Shamir [Sha84] 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) $\mathsf{Encrypt}$ allows for encrypting messages corresponding to an identity, and (4) $\mathsf{Decrypt}$ 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 [Coc01,BF01] 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. 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.[1]

The more functional notion of hierarchical IBE (HIBE) [HL02,GS02] 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

*Research supported in part from AFOSR YIP Award, DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, NSF CRII Award 1464397, and research grants by the Okawa Foundation, Visa Inc., and 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 postdoc fellowship of the German Academic Exchange Service (DAAD)

[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.

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.

The first IBE schemes were realized by Boneh and Franklin [BF01] and Cocks [Coc01]. Subsequently, significant research effort has been devoted to realizing IBE and HIBE schemes. By now, several constructions of IBE are known based on (i) various assumptions on groups with a bilinear map, e.g. [BF01, CHK03, BB04a, BB04b, Wat05, OT10], (ii) the quadratic residuocity assumption [Coc01, BGH07] (in the random oracle model [BR93]), or (iii) the learning-with-errors (LWE) assumption [GPV08, CHKP10, AB09]. On the other hand, HIBE schemes are known based on (i) various assumptions on groups with a bilinear map [HL02, GS02, BB04a, BBG05, GH09, Wat09, SW08, LW10], or (ii) LWE [CHKP10, ABB10a, ABB10b].

On the negative side, Boneh, Papakonstantinou, Rackoff, Vahlis, and Waters [BPR+08] show that IBE cannot be realized using trapdoor permutations or CCA-secure public-key encryption in a black-box manner. Furthermore, Papakonstantinou, Rackoff and Vahlis [PRV12] show that black-box use of a group over which DDH is assumed to be hard is insufficient for realizing IBE.

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). In the group of quadratic residues this problem is as hard as the Factoring problem [Shm85, McC88, BBR97]. Therefore, this implies a solution based on the hardness of factoring as well.

Our constructions bypass the known impossibility results [BPR+08, PRV12] 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 inefficient. In Section 6, we suggest ideas for reducing the non-black-box 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 Diffie-Hellman Assumption.

Subsequent work. In a followup paper [DG17] we show how the techniques from this paper can be used to obtain generic constructions of fully-secure IBE and selectively-secure HIBE starting with any selectively-secure IBE scheme.

2 Our Techniques

In this section, we give an intuitive explanation of our construction of IBE from the Decisional Diffie-Hellman (DDH) Assumption. We defer the details on constructing HIBE and obtaining the same results based on Computational Diffie-Hellman to the main body of the paper.

We start by describing a chameleon hash function [KR98] that supports certain encryption and decryption procedures. We refer to this new primitive as a *chameleon encryption scheme*.[2] Subsequently, we describe how chameleon encryption along with garbled circuits can be used to realize IBE.

[2]The notion of chameleon hashing is closely related to the notion of chameleon commitment scheme [BCC88] and we refer the reader to [KR98] for more discussion on this.

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 the chameleon hash function and then the associated encryption and decryption procedures. Recall that a chameleon hash function is a collision resistant hash function for which the knowledge of a trapdoor enables collision finding.

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 j^{th} bit of $\mathsf{x} \in \{0,1\}^n$. It is not very hard to note 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, 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 i^{th} bit of the preimage it can supply is equal to the value b chosen at the time of encryption. Our security requirement roughly is 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.[3]

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 ct where $\mathsf{ct} = (e, c, c', \{c_{j,0}, c_{j,1}\}_{j\in[n]\setminus\{i\}})$ and

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

[3] The success of decryption is conditioned on certain requirements placed on (x, r). This restricted decryption capability is reminiscent of the concepts of witness encryption [GGSW13] and extractable witness encryption [BCP14, ABG+13].

It is easy to see that if $\mathsf{x}_i = b$ then decryption $\mathsf{Dec}(\mathsf{ct}, (\mathsf{x}, r))$ can just output

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

However, if $\mathsf{x}_i \neq b$ then the decryptor has access to the value g^{ρ}_{i,x_i} but not $g^{\rho}_{i,b}$, and this prevents him from learning the message m. Formalizing this intuition, we can argue security of this scheme based on the DDH assumption.[4] In a bit more detail, we can use an adversary $\mathcal{A}$ breaking the security of the chameleon encryption 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 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 same distribution as in the security experiment, thus $\mathcal{A}$'s advantage in guessing m remains the same. On the other hand, 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.

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. This is the intuition behind the black-box impossibility results for realizing IBE based on trapdoor permutations and CCA secure encryption [BPR+08,PRV12]. At a very high level, we overcome this intuitive barrier by actually allowing for exponentially many public keys which are somehow compressed into small public parameters using our chameleon hash function. We start by describing how these keys are sampled and hashed.

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 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, in order to setup 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 $\mathsf{v} \in \{0,1\}^{\leq n-1} \cup \{\epsilon\}$[5]

[4] In Section 5, we explain our constructions of chameleon encryption based on the (Computational) Diffie-Hellman Assumption, or the Factoring Assumption.

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

the hash value $\mathsf{h_v}$ is set as:

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

where r_{v} for each $\mathsf{v} \in \{0,1\}^{<n} \cup \{\epsilon\}$ are chosen randomly.

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 $\mathsf{h_v}$ as $\mathsf{H}(\mathsf{k}_{|\mathsf{v}|}, 0^\lambda; \omega_{\mathsf{v}})$. 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 $\mathsf{h_v}$ indeed satisfies Equation 1. Furthermore, in order 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 can be represented succinctly.

What are the public parameters? Note that the root hash value h_ε somehow binds the entire tree of hash values. With this in mind, we sent 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 $\mathsf{dk_{id}}$.[6] Specifically, the secret key $\mathsf{sk_{id}}$ for identity id consists of $(\{\mathsf{lk_v}\}_{\mathsf{v}\in V}, \mathsf{dk_{id}})$ where $V := \{\varepsilon, \mathsf{id}[1], \ldots \mathsf{id}[1 \ldots n-1]\}$ and

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

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.[7] Chameleon encryption allows an encryptor knowing a key k and a hash value h to encrypt a set 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 $\{\mathsf{lab}_{j,\mathsf{x}_j}\}_j$. On the other hand, security of chameleon encryption guarantees that the receiver learns nothing about the remaining labels. In summary, using this mechanism, an 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 $\mathsf{sk_{id}} = (\{\mathsf{lk_v}\}_{\mathsf{v}\in V}, \mathsf{dk_{id}})$ will be able 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 $\mathsf{h_v}$ and generates chameleon encryptions of the input labels of the next garbled circuit $\tilde{P}^{i+1}$ using a $\mathsf{k}_{|\mathsf{v}|}$

[6] We note that our key generation mechanism can be seen as an instantiation of the Naor Yung [NY89] tree-based construction of signature schemes from universal one-way hash functions and one-time signatures. This connection becomes even more apparent in the follow up paper [DG17].

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

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 $\mathsf{pk}_{\mathsf{id}}$ 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 $i \in \{0 \dots n-1\}$ and $\mathsf{v} = \mathsf{id}[1 \dots i]$ 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. 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:

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

The garbled circuit has $\mathsf{id}[1]$ and the input labels $\{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}_{j\in[\lambda]}$ hardwired in it. 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 j^{th} 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 $\mathsf{lk}_{\mathsf{id}[1]}$ and $r_{\mathsf{id}[1]}$ 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 revealing an encryption of the message m under the encryption key $\mathsf{pk}_{\mathsf{id}}$. 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 [LO13, GHL+14, GLOS15, GLO15, CDG+17]. 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 $\tilde{P}^0, \dots \tilde{P}^{n-1}, \tilde{T}$ 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 $\mathsf{pk}_{\mathsf{id}^*}$ 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 of 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 $\mathsf{H}^{-1}(\mathsf{t}_0, (0^{2\lambda}, \omega_\varepsilon), \mathsf{h}_0 \| \mathsf{h}_1)$. We instead generate h_ε directly to be equal to the value r_ε are set 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 not generates 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 at this point the trapdoor t_ε is not being used at all and $\tilde{P}^0$ is the simulated form. 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

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

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 and the same sequence of hybrids can be repeated for the next garbled circuit $\tilde{P}^1$. The only change in this step is that now both h_0 and h_1 will be generated (if needed) by first sampling their children. The full proof of security is provided in Section 6.2.

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.[8] Alternatively, for any distribution D we use $x \xleftarrow{\$} D$ to mean that x is sampled from the distribution D. We use the operator $:=$ to represent assignment and $=$ to denote an equality check.

3.1 Computational Problems

Definition 3.1 (The Diffie-Hellman (DH) Problem)**.** *Let $(\mathbb{G}, \cdot)$ be a cyclic group of order p with generator g. Let a, b be sampled uniformly at random from $\mathbb{Z}_p$ (i.e., $a, b \xleftarrow{\$} \mathbb{Z}_p$). Given (g, g^a, g^b), the $\mathsf{DH}(\mathbb{G})$ problem asks to compute g^{ab}.*

Definition 3.2 (The Factoring Problem)**.** *Given a Blum integer $N = pq$ (p and q are large primes with $p = q = 3 \mod 4$) the FACT problem asks to compute p and q.*

3.2 Identity-Based Encryption

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

Definition 3.3 (Identity-Based Encryption (IBE) [Sha84, BF01])**.** *An* identity-based encryption *scheme consists of four PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *defined as follows:*

- $\mathsf{Setup}(1^\lambda)$*: given the security parameter, it outputs a master public key* mpk *and a master secret key* msk*.*

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

- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$*: given the master secret key* msk *and an identity* $\mathsf{id} \in \{0,1\}^n$*, it outputs a decryption key* $\mathsf{sk}_{\mathsf{id}}$*.*
- $\mathsf{Encrypt}(\mathsf{mpk}, \mathsf{id}, \mathsf{m})$*: 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:*

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

  where $IND^{IBE}_{\mathcal{A}}$ *is shown in Figure 1, and for each key query* id *that* $\mathcal{A}$ *sends to the* KeyGen *oracle, it must hold that* $\mathsf{id} \neq \mathsf{id}^*$*.*

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

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.*

Figure 1: The $\mathrm{IND}^{\mathrm{IBE}}_{\mathcal{A}}$ Experiment

Hierarchical Identity-Based Encryption (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 $\mathsf{sk}_{\mathsf{id}\|\mathsf{id}'}$.

Correctness condition for HIBE is same as it was from IBE. Additionally, the security property is analogous to $\text{IND}_{\mathcal{A}}^{\text{IBE}}(1^\lambda)$ 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 $\text{IND}_{\mathcal{A}}^{\text{HIBE}}$ is shown formally in Figure 2.

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

1. $(\mathsf{id}^*, \mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_1$ where $|m_0| = |m_1|$.
2. $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$.
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.

Figure 2: The $\text{IND}_{\mathcal{A}}^{\text{HIBE}}$ Experiment

3.3 Garbled Circuits

Garbled circuits were first introduced by Yao [Yao82] (see Lindell and Pinkas [LP09] and Bellare et al. [BHR12] for a detailed proof and further discussion). A circuit garbling scheme is a tuple of PPT algorithms $(\mathsf{GCircuit}, \mathsf{Eval})$. Very roughly $\mathsf{GCircuit}$ is the circuit garbling procedure and Eval the corresponding evaluation procedure. More formally:

- $(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,b}\}_{w \in \mathsf{inp}(\mathsf{C}), b \in \{0,1\}}) \xleftarrow{\$} \mathsf{GCircuit}\left(1^\lambda, \mathsf{C}\right)$: $\mathsf{GCircuit}$ takes as input a security parameter λ and a circuit C. This procedure outputs a *garbled circuit* $\widetilde{\mathsf{C}}$ and labels $\{\mathsf{lab}_{w,b}\}_{w \in \mathsf{inp}(\mathsf{C}), b \in \{0,1\}}$ where each $\mathsf{lab}_{w,b} \in \{0,1\}^\lambda$.[9]
- $y := \mathsf{Eval}\left(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w \in \mathsf{inp}(\mathsf{C})}\right)$: Given a garbled circuit $\widetilde{\mathsf{C}}$ and a garbled input represented as a sequence of input labels $\{\mathsf{lab}_{w,x_w}\}_{w \in \mathsf{inp}(\mathsf{C})}$, 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:

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

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

[9] 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.

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

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

where $(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,b}\}_{w\in\mathsf{inp}(\mathsf{C}),b\in\{0,1\}}) \xleftarrow{\$} \mathsf{GCircuit}\left(1^\lambda, \mathsf{C}\right)$.[10]

4 Chameleon Encryption

In this section, we give the definition of a chameleon encryption scheme.

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*.*
- $\mathsf{H}(\mathsf{k}, \mathsf{x}; r)$*: 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*.*[11]
- $\mathsf{Dec}(\mathsf{k}, (\mathsf{x}, r), \mathsf{ct})$*: 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[12]

- ***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 it holds that*

  $$\mathsf{H}(\mathsf{k}, \mathsf{x}; r) = \mathsf{H}(\mathsf{k}, \mathsf{x}'; r'),$$

  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* $\mathsf{Dec}(\mathsf{k}, \mathsf{ct}, (\mathsf{x}, r)) = \mathsf{m}$*.*

[10] 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 $\tilde{C}$.

[11] ct is assumed to contain (h, i, b).

[12] 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.

- ***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:*

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

where $IND^{CE}_{\mathcal{A}}$ *is shown in Figure 3.*

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

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.*

Figure 3: The $\mathrm{IND}^{\mathrm{CE}}_{\mathcal{A}}$ Experiment

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 (not necessarily prime) is the order of $\mathbb{G}$.[13] We will now describe a chameleon encryption scheme assuming that the $\mathsf{DH}(\mathbb{G})$ problem is hard.

- $\mathsf{Gen}(1^\lambda, n)$: For each $j \in [n]$, choose uniformly random values $\alpha_{j,0}, \alpha_{j,1} \xleftarrow{\$} \mathsf{Sample}(\mathbb{G})$ and compute $g_{j,0} := g^{\alpha_{j,0}}$ and $g_{j,1} := g^{\alpha_{j,1}}$. Output (k, t) where[14]

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

- $\mathsf{H}(\mathsf{k}, \mathsf{x}; r)$: Parse k as in Equation 2, sample $r \xleftarrow{\$} \mathsf{Sample}(\mathbb{G})$, 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'.

- $\mathsf{Enc}(\mathsf{k}, (\mathsf{h}, i, b), \mathsf{m})$: Parse k as in Equation 2, $\mathsf{h} \in \mathbb{G}$ and $\mathsf{m} \in \{0,1\}$. Sample $\rho \xleftarrow{\$} \mathsf{Sample}(\mathbb{G})$ and proceed as follows:

[13] We will later provide instantiations of $\mathbb{G}$ which are of prime order and composite order. The use of $\mathsf{Sample}(\mathbb{G})$ procedure is done to unify these two instantiations.

[14] We also implicitly include the public and secret parameters for the group $\mathbb{G}$ in k and t respectively.

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 $e := \mathsf{m} \oplus \mathsf{HardCore}(g_{i,b}^{\rho})$.[15]
5. Output $\mathsf{ct} := \left(e, c, c', \begin{pmatrix} c_{1,0}, c_{2,0} \ldots, c_{n,0} \\ c_{1,1}, c_{2,1}, \ldots, c_{n,1} \end{pmatrix}\right)$.

- $\mathsf{Dec}(\mathsf{k}, (\mathsf{x}, r), \mathsf{ct})$: Parse $\mathsf{ct} = \left(e, c, c', \begin{pmatrix} c_{1,0}, c_{2,0} \ldots, c_{n,0} \\ c_{1,1}, c_{2,1}, \ldots, c_{n,1} \end{pmatrix}\right)$

  Output $e \oplus \mathsf{HardCore}\left(\frac{c'}{c^r \cdot \prod_{j \in [n] \backslash \{i\}} c_{j,\mathsf{x}_j}}\right)$.

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* $\mathsf{DH}(\mathbb{G})$ *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 and we that below.

- **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 can be established as follows.

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

[15] We assume that the $\mathsf{HardCore}(g^{ab})$ 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-Levin [GL89] construction. We skip the details of that with the goal of keeping exposition simple.

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 $\mathsf{ct} = \left(e, c, c', \begin{pmatrix} c_{1,0}, c_{2,0}\ldots, c_{n,0} \\ c_{1,1}, c_{2,1}, \ldots, c_{n,1}\end{pmatrix}\right)$ 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

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

Now we will provide a PPT reduction $\mathcal{R}^{\mathcal{A}}$ which 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, **Reduction** $\mathcal{R}^{\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)}(g,U,V)$ 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 $x \xleftarrow{\$} \{0,1\}$ and $i^* \xleftarrow{\$} [n]$ and reassign $g_{i^*,x} := U$. 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 $\mathsf{x}_i = x$ 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 $\mathsf{ct} := \left(e, c, c', \begin{pmatrix} c_{1,0}, c_{2,0} \ldots, c_{n,0} \\ c_{1,1}, c_{2,1}, \ldots, c_{n,1} \end{pmatrix}\right)$ 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 $x_i \neq x$. Now observe that the distribution of k in Step 3 is statistically close to distribution resulting from Gen. This implies that (1) the view of the attacker in Step 3 is statistically close to experiment $\mathrm{IND}^{\mathrm{CE}}_{\mathcal{A}}$, 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 $\mathrm{IND}^{\mathrm{CE}}_{\mathcal{A}}$ 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 $\frac{1}{2} \cdot \left(1 - \frac{1}{2n}\right) + \frac{1}{2n} \cdot \left(\frac{1}{2} + \mu\right) = \frac{1}{2} + \frac{\mu}{2n}$ 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 $\mathbb{G}$ where $\mathsf{DH}(\mathbb{G})$ is assumed to be hard. Candidates are prime order multiplicative subgroups of finite fields [DH76] and elliptic curve groups [Mil86, Kob87].

Corollary 5.2. *Under the assumption that* $\mathsf{DH}(\mathbb{G})$ *is hard over some group* $\mathbb{G}$*, there exists a chameleon encryption scheme.*

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[16] with $P = Q = 3 \mod 4$). Let g be a random generator of $\mathbb{G}$ and $\mathsf{Sample}(\mathbb{G})$ just outputs a uniformly random number from the set $[(N-1)/4]$. Shmuely [Shm85] and McCurley [McC88] proved that the $\mathsf{DH}(\mathbb{QR}_N)$ problem is at least as hard as FACT (also see [BBR97, HK09]).

For this instantiation, we assume that the Gen algorithm generates a fresh Blum integer $N = PQ = (2p+1)(2q+1)$, 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 $\mathsf{Sample}(\mathbb{G})$.

Hence, using the group $\mathbb{QR}_N$ in the above described construction yields a construction of chameleon encryption based on the FACT Assumption.

Corollary 5.3. *Under the assumption that* FACT *is hard there exists a chameleon encryption scheme.*

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

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 encryption scheme and $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ be any semantically secure public-key encryption scheme.[17] We let $\mathsf{id}[i]$ denote the i^{th}-bit of id and let $\mathsf{id}[1 \dots i]$ denote the first i bits of id. Note that $\mathsf{id}[1 \dots 0]$ is the empty string denoted by ε of length 0.

NodeGen and LeafGen functions. As explained in the introduction, we need an exponential sized tree of hash values. The functions NodeGen and LeafGen provides efficient access to the *hash value* corresponding to any node in this (exponential sized) tree. We will use these function 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 $\mathsf{v} \in \{0,1\}^{\leq n-2} \cup \{\varepsilon\}$. On the other hand, 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 4.

$\mathsf{NodeGen}((\mathsf{k}_0, \dots \mathsf{k}_{n-1}), (\mathsf{t}_0, \dots \mathsf{t}_{n-1}, s), \mathsf{v})$:

1. Let $i := |\mathsf{v}|$ (length of v) and generate
$$\begin{aligned} \mathsf{h}_\mathsf{v} &:= \mathsf{H}(\mathsf{k}_i, 0^{2\lambda}; \mathsf{PRF}(s, \mathsf{v})), \\ \mathsf{h}_{\mathsf{v}\|0} &:= \mathsf{H}(\mathsf{k}_{i+1}, 0^{2\lambda}; \mathsf{PRF}(s, \mathsf{v}\|0)), \\ \mathsf{h}_{\mathsf{v}\|1} &:= \mathsf{H}(\mathsf{k}_{i+1}, 0^{2\lambda}; \mathsf{PRF}(s, \mathsf{v}\|1)). \end{aligned}$$
2. $r_\mathsf{v} := \mathsf{H}^{-1}(\mathsf{t}_\mathsf{v}, (0^{2\lambda}, \mathsf{PRF}(s, \mathsf{v})), \mathsf{h}_{\mathsf{v}\|0} \| \mathsf{h}_{\mathsf{v}\|1})$.
3. Output $(\mathsf{h}_\mathsf{v}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|1}, r_\mathsf{v})$.

$\mathsf{LeafGen}(\mathsf{k}_{n-1}, (\mathsf{t}_{n-1}, s), \mathsf{v})$:

1. Generate
$$\begin{aligned} \mathsf{h}_\mathsf{v} &:= \mathsf{H}(\mathsf{k}_{n-1}, 0^{2\lambda}; \mathsf{PRF}(s, \mathsf{v})) \\ (\mathsf{ek}_{\mathsf{v}\|0}, \mathsf{dk}_{\mathsf{v}\|0}) &:= \mathsf{G}(1^\lambda; \mathsf{PRF}(s, \mathsf{v}\|0)), \\ (\mathsf{ek}_{\mathsf{v}\|1}, \mathsf{dk}_{\mathsf{v}\|1}) &:= \mathsf{G}(1^\lambda; \mathsf{PRF}(s, \mathsf{v}\|1)). \end{aligned}$$
2. $r_\mathsf{v} := \mathsf{H}^{-1}(\mathsf{t}_n, (0^{2\lambda}, \mathsf{PRF}(s, \mathsf{v})), \mathsf{ek}_{\mathsf{v}\|0} \| \mathsf{ek}_{\mathsf{v}\|1})$.
3. Output $((\mathsf{h}_\mathsf{v}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1}, r_\mathsf{v}), \mathsf{dk}_{\mathsf{v}\|0}, \mathsf{dk}_{\mathsf{v}\|1})$.

Figure 4: Description of NodeGen and LeafGen.

Construction. We describe our IBE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$.

- $\mathsf{Setup}(1^\lambda, 1^n)$: Proceed as[18] follows:
  1. Sample $s \xleftarrow{\$} \{0,1\}^\lambda$ (seeds 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 $(\mathsf{h}_\varepsilon, \mathsf{h}_0, \mathsf{h}_1, r_\varepsilon) := \mathsf{NodeGen}((\mathsf{k}_0, \dots \mathsf{k}_{n-1}), (\mathsf{t}_0, \dots \mathsf{t}_{n-1}, s), \varepsilon)$

[17] 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 $\mathsf{E}(\mathsf{ek}, \mathsf{m})$ takes as input an encryption key ek and a message m and outputs a ciphertext ct. Finally, the decryption algorithm $\mathsf{D}(\mathsf{dk}, \mathsf{ct})$ takes as input the secret key and the ciphertext and outputs the encrypted message m.

[18] 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.

4. Output $(\mathsf{mpk}, \mathsf{msk})$ where $\mathsf{mpk} := (\mathsf{k}_0, \ldots \mathsf{k}_{n-1}, \mathsf{h}_\varepsilon)$ and $\mathsf{msk} := (\mathsf{mpk}, \mathsf{t}_0, \ldots \mathsf{t}_{n-1}, s)$

- $\mathsf{KeyGen}(\mathsf{msk} = ((\mathsf{k}_0, \ldots \mathsf{k}_{n-1}, \mathsf{h}_\varepsilon), \mathsf{t}_0, \ldots \mathsf{t}_{n-1}, s), \mathsf{id} \in \{0,1\}^n)$:

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

- $\mathsf{Encrypt}(\mathsf{mpk} = (\mathsf{k}_0, \ldots \mathsf{k}_{n-1}, \mathsf{h}_\varepsilon), \mathsf{id} \in \{0,1\}^n, \mathsf{m})$: Before describing the encryption procedure we describe two circuits[19] that will be garbled during the encryption process.

  – $\mathsf{T}[\mathsf{m}](\mathsf{ek})$: 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 j^{th} 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 $\mathsf{m} := \mathsf{Dec}(\mathsf{dk}_\mathsf{id}, f)$.

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

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 optimization 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 a correctly generated ciphertext on decryption 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 $\mathsf{v} = \mathsf{id}[1 \ldots n-1]$ 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 $\mathsf{ct} = (\{\mathsf{lab}_{j,\mathsf{h}_{\varepsilon,j}}\}_{j \in [\lambda]}, \{\tilde{P}^0, \ldots, \tilde{P}^{n-1}, \tilde{T}\})$. 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 $e_{j,b}$ which 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 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

We are now ready to prove the security of the IBE construction above. For the sake of contradiction we proceed by assuming that there exists an adversary $\mathcal{A}$ such that $\Pr[\mathrm{IND}^{\mathrm{IBE}}_{\mathcal{A}}(1^\lambda) = 1] \geq \frac{1}{2} + \epsilon$ for a non-negligible ϵ (in λ), where $\mathrm{IND}^{\mathrm{IBE}}_{\mathcal{A}}$ is shown in Figure 1. 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 consist of a sequence of $n+2$ hybrids $\mathcal{H}_{-1}, \mathcal{H}_0, \mathcal{H}_1, \ldots \mathcal{H}_{n+1}$. We next describe these hybrids .

- $\mathcal{H}_{-1}$: This hybrid corresponds to the experiment $\text{IND}^{\text{IBE}}_{\mathcal{A}}$ as shown in Figure 1.

- $\mathcal{H}_0$: 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 in there unsimulated versions. 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 $i = n-1, \dots, \tau$ 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 $i = \tau - 1, \dots, 0$, set $\mathsf{v} = \mathsf{id}^*[1 \dots i-1]$ 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 $\mathsf{ct} := (\{\mathsf{lab}_{j,\mathsf{h}_{\varepsilon,j}}\}_{j\in[\lambda]}, \{\tilde{P}^0, \dots, \tilde{P}^{n-1}, \tilde{T}\})$ where $\mathsf{h}_{\varepsilon,j}$ is the j^{th} bit of h_ε.

  The computational indistinguishability between hybrids $\mathcal{H}_{\tau-1}$ and $\mathcal{H}_\tau$ is based on Lemma 6.1 which is proved in Section 6.3.

  Lemma 6.1. *For each $\tau \in \{1 \dots n\}$ 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 $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ 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 $j^* = 0$, 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, it aborts and outputs a random bit. Otherwise, if its guess turns out to be correct, which happens with probability at least $1/q$, 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 6.1

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 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 $\mathcal{H}_{\tau,0}$, 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,

$$\mathsf{h}_\mathsf{v} := \begin{cases} \mathsf{H}(\mathsf{k}_\tau, \mathsf{h}_{\mathsf{v}\|0} \| \mathsf{h}_{\mathsf{v}\|1}; r_\mathsf{v}) & \text{if } \tau < n-1 \\ \mathsf{H}(\mathsf{k}_\tau, \mathsf{ek}_{\mathsf{v}\|0} \| \mathsf{ek}_{\mathsf{v}\|1}; r_\mathsf{v}) & \text{otherwise} \end{cases}.$$

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 $\mathsf{v} = \mathsf{id}^*[1 \dots \tau]$ and recall that

$$\mathsf{lk}_\mathsf{v} = \begin{cases} (\mathsf{h}_\mathsf{v}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|1}, r_\mathsf{v}) & \text{if } \tau < n-1 \\ (\mathsf{h}_\mathsf{v}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1}, r_\mathsf{v}) & \text{if } \tau = n-1 \end{cases}.$$

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

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

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

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 $\{\mathsf{Enc}(\mathsf{k}_\mathsf{v}, (\mathsf{h}_\mathsf{v}, j, b), \mathsf{lab}_{j,b})\}_{j\in[\lambda], b\in\{0,1\}}$ is exactly the output of the circuit $\mathsf{P}[\mathsf{id}[\tau+1], \mathsf{k}_\tau, \overline{\mathsf{lab}}]$ on input h_v. On the other hand, 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 $\mathsf{v} := \mathsf{id}[1 \dots \tau]$ we change

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

to

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

Notice that 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 will now 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: We note that 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 $\mathsf{NodeGen}$ and $\mathsf{LeafGen}$.

  Computational 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 and $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ be any semantically secure public-key encryption scheme. We let $\mathsf{id}[i]$ denote the i^{th}-bit of id and $\mathsf{id}[1\ldots i]$ denote the first i bits of id (and $\mathsf{id}[1\ldots 0] = \varepsilon$).

Notation for the pseudorandom function F. Let $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ be a length tripling pseudorandom generator and $\mathsf{PRG}_0, \mathsf{PRG}_1$ and PRG_2 be the $1\ldots\lambda$, $\lambda+1\ldots 2\lambda$ and $2\lambda+1\ldots 3\lambda$ bits of the output of PRG, respectively. Now define a GGM-type [GGM84] pseudo-random function $\mathsf{F} : \{0,1\}^\lambda \times \{0,1,2\}^* \to \{0,1\}^\lambda$ such that $\mathsf{F}(s,x) := \mathsf{PRG}_{x_n}(\mathsf{PRG}_{x_{n-1}}(\ldots(\mathsf{PRG}_{x_1}(s))\ldots))$, where $n = |x|$ and for each $i \in [n]$ x_i is the i^{th} element (from 0, 1 or 2) of string x.[20]

$\mathsf{NodeGen}$ and $\mathsf{NodeGen}'$ functions. As explained in the introduction, we need an exponential sized tree of local-keys. The function $\mathsf{NodeGen}$ provides efficient access to *local-keys* corresponding to any node in this (exponential sized) tree. We will use this function repeatedly in our construction.

[20] $\mathsf{F}(s,\varepsilon)$ is set to output s.

The function takes as input the 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 $s = (s_1, s_2, s_3)$ seeds for the pseudo-random function PRF. This function is explained in the Figure 5.

$\mathsf{NodeGen}(\mathsf{k}_G, \mathsf{v}, (s_1, s_2, s_3))$:

1. Obtain ω_1, ω_2, and ω_3 be the first, second and third $\lambda/3$ bits of s_1, respectively.
2. Generate $(\mathsf{k}_\mathsf{v}, \mathsf{t}_\mathsf{v}) := \mathsf{Gen}(1^\lambda; \omega_1)$ 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 seed s_2 and $\mathsf{k}_{\mathsf{v}\|1}, \mathsf{h}_{\mathsf{v}\|1}$ using seed s_3.
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

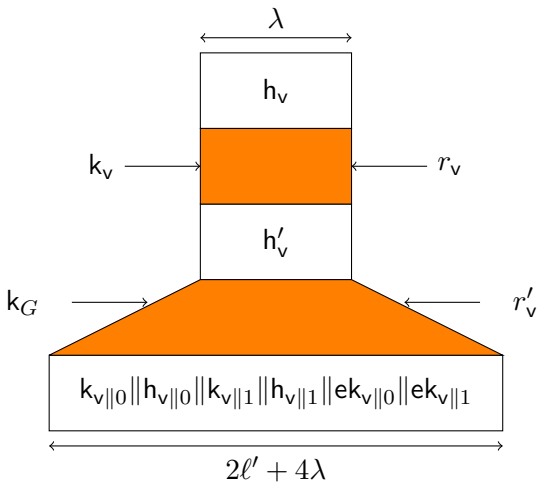


Figure 5: Explanation on how $\mathsf{NodeGen}$ works. Strings ω_1, ω_2 and ω_3 are used as randomness for cryptographic functions and can be sufficiently expanded using a PRG.

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

Construction. We describe our HIBE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$.

- $\mathsf{Setup}(1^\lambda)$: Proceed as follows:
  1. Sample $s \xleftarrow{\$} \{0,1\}^\lambda$ (seeds for the pseudorandom function PRF).
  2. Setup a global hash function $(\mathsf{k}_G, \cdot) := \mathsf{Gen}(1^\lambda, 2\ell + 2\lambda)$[21] where $\ell = \ell' + \lambda$ and ℓ' is the length of k generated from $\mathsf{Gen}(1^\lambda, \lambda)$.
  3. Obtain $(\mathsf{k}_\varepsilon, \mathsf{h}_\varepsilon, r_\varepsilon, \mathsf{h}'_\varepsilon, r'_\varepsilon, \mathsf{k}_0, \mathsf{h}_0, \mathsf{k}_1, \mathsf{h}_1) := \mathsf{NodeGen}(\mathsf{k}_G, \varepsilon, s)$
  4. Output $(\mathsf{mpk}, \mathsf{msk})$ where $\mathsf{mpk} := (\mathsf{k}_G, \mathsf{k}_\varepsilon, \mathsf{h}_\varepsilon)$ and $\mathsf{msk} = \mathsf{sk}_\varepsilon := (\varepsilon, \emptyset, s, \perp)$
- $\mathsf{KeyGen}(\mathsf{sk}_\mathsf{id} = (\mathsf{id}, \{\mathsf{lk}_\mathsf{v}\}_{\mathsf{v} \in V}, s, \mathsf{dk}_\mathsf{id}), \mathsf{id}' \in \{0,1\}^*)$:[22]

[21] The trapdoor for the global hash function is not needed in the construction or the proof and is therefore dropped.

[22] HIBE is often defined to have separate KeyGen and $\mathsf{Delegate}$ 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.

Let $n := |\mathsf{id}'|$ and set $V' := \{\mathsf{id}\|\mathsf{id}'[1 \dots j-1]\}_{j\in[n]}$
For all $\mathsf{v} \in V'$:
$\mathsf{lk}_{\mathsf{v}} := \mathsf{NodeGen}(\mathsf{k}_G, \mathsf{v}, (\mathsf{F}(s, \mathsf{v}\|2), \mathsf{F}(s, \mathsf{v}\|0\|2), \mathsf{F}(s, \mathsf{v}\|1\|2)))$
Let $\mathsf{v} := \mathsf{id}\|\mathsf{id}'[1 \dots n-1]$
$\mathsf{dk}_{\mathsf{id}\|\mathsf{id}'} := \mathsf{NodeGen}'(\mathsf{k}_G, \mathsf{v}, (\mathsf{F}(s, \mathsf{v}\|2), \mathsf{F}(s, \mathsf{v}\|0\|2), \mathsf{F}(s, \mathsf{v}\|1\|2)), \mathsf{id}'[n])$
Output $\mathsf{sk}_{\mathsf{id}\|\mathsf{id}'} := (\mathsf{id}, \{\mathsf{lk}_{\mathsf{v}}\}_{\mathsf{v}\in V\cup V'}, \mathsf{F}(s, \mathsf{id}'), \mathsf{dk}_{\mathsf{id}\|\mathsf{id}'})$

Remark: We note that in our construction the secret key for any identity is unique regardless of 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 $\mathsf{E}(\mathsf{ek}, \mathsf{m})$.
  - $\mathsf{Q}_{last}[\beta \in \{0,1\}, \mathsf{k}_G, \overline{\mathsf{tlab}}](\mathsf{h})$: 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 short for $\{\mathsf{qlab}_{j,b}\}_{j\in[\lambda],b\in\{0,1\}}$.

  Encryption proceeds as follows:

  1. Compute $\tilde{T}$ as:
  $$(\tilde{T}, \overline{\mathsf{tlab}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{Q}_{out}[\mathsf{k}_G, \mathsf{m}])$$
  2. For $i = n, \dots, 1$ 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 j^{th} bit of x_ε.

- $\mathsf{Decrypt}(\mathsf{ct}, \mathsf{sk}_{\mathsf{id}} = (\mathsf{id}, \{\mathsf{lk}_{\mathsf{v}}\}_{\mathsf{v}\in V}), s, \mathsf{dk}_{\mathsf{id}})$: 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 j^{th} 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 $V = \{\mathsf{id}[1 \dots j-1]\}_{j\in[n]}$.)
  3. For each $i \in [n]$, proceed as follows:

(a) Set $\mathsf{v} := \mathsf{id}[1 \dots i-1]$, $x_{\mathsf{v}} := \mathsf{k}_{\mathsf{v}} \| \mathsf{h}_{\mathsf{v}}$, $y_{\mathsf{v}} := \mathsf{h}'_{\mathsf{v}}$, 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 $z_{\mathsf{v}} := \mathsf{ek}_{\mathsf{id}}$.[23]

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

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-leaf path corresponding to identity id. Then the secret key $\mathsf{sk}_{\mathsf{id}}$ consists of $\{\mathsf{lk}_{\mathsf{v}}\}_{\mathsf{v}\in V}$, $\mathsf{dk}_{\mathsf{id}}$ and a seed of the pseudorandom function F. $\{\mathsf{lk}_{\mathsf{v}}\}_{\mathsf{v}\in V}$, $\mathsf{dk}_{\mathsf{id}}$ and will be used for decryption and s is used for delegating keys. Note that by construction (and the trapdoor collision property of the chameleon encryption scheme for the first equation below) for all nodes $\mathsf{v} \in V$ we have that:

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

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'

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

This fact follows directly from the the following property of the GGM PRF. Specifically, for every x we have that $\mathsf{F}(s, \mathsf{id} \| x) = \mathsf{F}(\mathsf{F}(s, \mathsf{id}), x)$.

[23] For $i < n$, z_{v} will become the x_{v} in next iteration.

7.2 Proof of Security

We are now ready to prove the selective security of the HIBE construction above. For the sake of contradiction we proceed by assuming that there exists an adversary $\mathcal{A}$ such that $\Pr[\mathrm{IND}^{\mathrm{HIBE}}_{\mathcal{A}}(1^\lambda) = 1] \geq \frac{1}{2} + \epsilon$ for a non-negligible ϵ (in λ), where $\mathrm{IND}^{\mathrm{HIBE}}_{\mathcal{A}}$ is shown in Figure 2. 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 the ciphertext. We describe our hybrids next. Our proof consist of a sequence of hybrids $\mathcal{H}_{-3}, \mathcal{H}_{-2}, \mathcal{H}_{-1}, \mathcal{H}_0, \mathcal{H}_1, \ldots \mathcal{H}_{n+2}$. 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 $\{\varepsilon, \mathsf{id}^*[1] \ldots \mathsf{id}^*[1 \ldots n-1]\}$.

- $\mathcal{H}_{-3}$: This hybrid corresponds to the experiment $\mathrm{IND}^{\mathrm{HIBE}}_{\mathcal{A}}$ as shown in Figure 2.
- $\mathcal{H}_{-2}$: In this hybrid, we change how the seed s of generated in Step 1 of Setup is used. Specifically, we sample $s \xleftarrow{\$} \{0,1\}^\lambda$ and generate

  1. For each $i \in [n]$, let $a_i := \mathsf{F}(s, \mathsf{id}^*[1 \ldots i-1] \| (1 - \mathsf{id}^*[i])$.
  2. $b := \mathsf{F}(s, \mathsf{id}^*)$.
  3. For each $i \in \{0 \ldots n-1\}$, let $c_i := \mathsf{F}(s, \mathsf{id}^*[1 \ldots i] \| 2)$.

  Now, through out the execution of the experiment we replace the use of s with the values $(\{a_i\}, b, \{c_i\})$. First, observe that (by standard properties of the GGM pseudorandom function) given these values we can generate $\mathsf{F}(s, \mathsf{v} \| 2)$ for all $\mathsf{v} \in \{0,1\}^* \cup \{\varepsilon\}$. Also, note that for the execution of the functions $\mathsf{NodeGen}$ and $\mathsf{NodeGen}'$ only $\mathsf{F}(s, \mathsf{v} \| 2)$ needs to be generated. Therefore, all executions of $\mathsf{NodeGen}$ and $\mathsf{NodeGen}'$ remain unaffected.

  Secondly, note that the $\mathcal{A}$ is only allowed to make KeyGen queries for identities $\mathsf{id} \notin V^* \cup \{\mathsf{id}^*\}$. Therefore, in order to answer these queries the experiment needs to generate $\mathsf{F}(s, \mathsf{v})$ for $\mathsf{v} \notin V^* \cup \{\mathsf{id}^*\}$. Observe that using $(\{a_i\}, b)$ by standard properties of the GGM pseudorandom function the experiment can compute $\mathsf{F}(s, \mathsf{v})$ for any $\mathsf{v} \notin V^*$. Therefore, all of $\mathcal{A}$'s KeyGen queries can be answered.[24]

  The hybrids $\mathcal{H}_{-3}$ and $\mathcal{H}_{-2}$ are the same distribution and the only change we have made is syntactic.

- $\mathcal{H}_{-1}$: In this hybrids, we change how each c_i is generated. In particular, we sample each c_i uniformly and independently instead of using F.

  The indistinguishability between hybrids $\mathcal{H}_{-2}$ and $\mathcal{H}_{-1}$ follows based on the pseudorandomness of the pseudorandom function F.

- $\mathcal{H}_0$: In this hybrid, we change how $\mathsf{NodeGen}$ and $\mathsf{NodeGen}'$ behave when computed with an input $\mathsf{v} \in V^*$.[25] For all $\mathsf{v} \notin V^*$ the behavior of $\mathsf{NodeGen}$ and $\mathsf{NodeGen}'$ remains unchanged.

[24] The experiment can provide $\mathsf{F}(s, \mathsf{id}^*)$ even though it does not appear in any of the $\mathcal{A}$'s secret key queries. The reason is that $\mathsf{F}(s, \mathsf{id}^*)$ allows the capabilities of delegation but not decryption for ciphertexts to identity id^*.

[25] Observe that these are specifically the cases in which one or two of the values s_1, s_2 and s_3 given as input to $\mathsf{NodeGen}$ and $\mathsf{NodeGen}'$ depend on the $\{c_i\}$ values.

At a high level, the goal is to change the generating of $\{\mathsf{lk_v}\}_{\mathsf{v}\in V^*}$ such that the trapdoor values $\mathsf{t_{v\in V^*}}$ are unused and so that the encryption key $\mathsf{ek_{id^*}}$ is sampled independent of everything else. The execution of $\mathsf{NodeGen}$ and $\mathsf{NodeGen'}$ for every $\mathsf{v} \notin V^*$ remain unaffected. In particular, at Setup time we proceed as follows and fix the values $\{\mathsf{lk_v}\}_{\mathsf{v}\in V^*}$ and $\{\mathsf{dk_{v\|0}}, \mathsf{dk_{v\|1}}\}_{\mathsf{v}\in V^*}$.[26]

1. For every $\mathsf{v} \in V^*$:
   (a) Generate $(\mathsf{k_v}, \mathsf{t_v}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$.
   (b) Generate $(\mathsf{ek_{v\|0}}, \mathsf{dk_{v\|0}}) \xleftarrow{\$} \mathsf{G}(1^\lambda)$ and $(\mathsf{ek_{v\|1}}, \mathsf{dk_{v\|1}}) \xleftarrow{\$} \mathsf{G}(1^\lambda)$.
   (c) Sample r'_v, r_v.
2. Let $S^* := \{\mathsf{id}^*[1\dots i-1]\|(1-\mathsf{id}^*[i])\}_{i\in[n]} \cup \{\mathsf{id}^*\}$. (Note that $S^* \cap V^* = \emptyset$.)
3. For all $\mathsf{v} \in S^*$ set $\mathsf{k_v}, \mathsf{h_v}$ as first two outputs of $\mathsf{NodeGen}(\mathsf{k}_G, \mathsf{v}, (\mathsf{F}(s, \mathsf{v}\|2), \mathsf{F}(s, \mathsf{v}\|0\|2), \mathsf{F}(s, \mathsf{v}\|1\|2)))$.
4. For each $i \in \{n-1\dots 0\}$:
   (a) Set $\mathsf{v} := \mathsf{id}^*[1\dots i]$
   (b) Generate $\mathsf{h'_v} := \mathsf{H}(\mathsf{k}_G, \mathsf{k_{v\|0}}||\mathsf{h_{v\|0}}||\mathsf{k_{v\|1}}||\mathsf{h_{v\|1}}\|\mathsf{ek_{v\|0}}\|\mathsf{ek_{v\|1}}; r'_\mathsf{v})$.
   (c) $\mathsf{h_v} := \mathsf{H}(\mathsf{k_v}, \mathsf{h'_v}; r_\mathsf{v})$.
   (d) $\mathsf{lk_v} := (\mathsf{k_v}, \mathsf{h_v}, r_\mathsf{v}, \mathsf{h'_v}, r'_\mathsf{v}, \mathsf{k_{v\|0}}, \mathsf{h_{v\|0}}, \mathsf{k_{v\|1}}, \mathsf{h_{v\|1}}, \mathsf{ek_{v\|0}}, \mathsf{ek_{v\|1}})$.
5. Output $\{\mathsf{lk_v}\}_{\mathsf{v}\in V^*}$ and $\{\mathsf{dk_{v\|0}}, \mathsf{dk_{v\|1}}\}_{\mathsf{v}\in V^*}$.

Statistical indistinguishability of hybrids $\mathcal{H}_{\tau,-1}$ and $\mathcal{H}_{\tau,0}$ follows from the trapdoor collision and uniformity properties of the chameleon encryption scheme. Note that in this hybrid the trapdoor $\mathsf{t_v}$ for any node $\mathsf{v} \in V^*$ is no longer being used.

- $\mathcal{H}_\tau$ for $\tau \in \{1\dots n\}$: This hybrid is identical to $\mathcal{H}_0$ except we change how the 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, $\tilde{P}^1, \tilde{Q}^1 \dots \tilde{P}^\tau, \tilde{Q}^\tau$) 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 $\mathsf{NodeGen}$. 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:
  $$(\tilde{T}, \overline{\mathsf{tlab}}) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{Q}_{out}[\mathsf{k}_G, \mathsf{m}])$$
  2. For $i = n, \dots, \tau+1$ 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
     $$(\tilde{Q}^i, \overline{\mathsf{qlab}}^i) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, \mathsf{Q}[\mathsf{id}[i], \mathsf{k}_G, \overline{\mathsf{plab}}^{i+1}]).$$

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

(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 j^{th} bit of x_ε.

The computational indistinguishability between hybrids $\mathcal{H}_{\tau-1}$ and $\mathcal{H}_\tau$ is based on Lemma 7.1 which is proved in Section 7.3.

Lemma 7.1. *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 using the garbling simulator. More specifically, instead of generating $\tilde{T}$ as

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

  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 gabled circuits.

- $\mathcal{H}_{n+2}$: 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 $\mathsf{E}(\mathsf{ek}_{\mathsf{id}^*}, 0)$.

  Note that the adversary $\mathcal{A}$ never queries for $\mathsf{sk}_{\mathsf{id}^*}$. 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 7.1

The proof follows 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 $\mathsf{v} = \mathsf{id}^*[1 \dots \tau - 1]$ and $\mathsf{lk}_{\mathsf{v}} = (\mathsf{k}_{\mathsf{v}}, \mathsf{h}_{\mathsf{v}}, r_{\mathsf{v}}, \mathsf{h}'_{\mathsf{v}}, r'_{\mathsf{v}}, \mathsf{k}_{\mathsf{v}\|0}, \mathsf{h}_{\mathsf{v}\|0}, \mathsf{k}_{\mathsf{v}\|1}, \mathsf{h}_{\mathsf{v}\|1}, \mathsf{ek}_{\mathsf{v}\|0}, \mathsf{ek}_{\mathsf{v}\|1})$ 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 j^{th} bit of x_{v}. Next, we set $\overline{\mathsf{plab}}^{i} := \{\mathsf{plab}^{i}_{j,x_{\mathsf{v},j}}, \mathsf{plab}^{i}_{j,x_{\mathsf{v},j}}\}_{j\in[\ell]}$.

  Computational indistinguishability of hybrids $\mathcal{H}_{\tau,0}$ and $\mathcal{H}_{\tau,1}$ follows by the security of the garbling scheme $\mathsf{GCircuit}$ 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 $\mathsf{v} = \mathsf{id}^*[1 \dots \tau - 1]$ 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 that 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^2 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 6.1.

  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 $\mathsf{v} = \mathsf{id}^*[1 \dots \tau - 1]$ 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 $z_{\mathsf{v}} := \mathsf{ek}_{\mathsf{id}^*}$. On the other hand, 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) otherwise 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 $\mathcal{H}_\tau$.

8 Acknowledgments

We thank the anonymous reviewers of CRYPTO 2017 for their valuable feedback.

References

[AB09] Shweta Agrawal and Xavier Boyen. Identity-based encryption from lattices in the standard model. Manuscript, 2009. http://www.cs.stanford.edu/ xb/ab09/.

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 98–115, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

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

[BB04a] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[BB04b] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 443–459, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

[BBR97] Eli Biham, Dan Boneh, and Omer Reingold. Generalized Diffie-Hellman modulo a composite is not weaker than factoring. Cryptology ePrint Archive, Report 1997/014, 1997. http://eprint.iacr.org/1997/014.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, October 1988.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 52–73, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BGH07] Dan Boneh, Craig Gentry, and Michael Hamburg. Space-efficient identity based encryption without pairings. In *48th Annual Symposium on Foundations of Computer Science*, pages 647–657, Providence, RI, USA, October 20–23, 2007. IEEE Computer Society Press.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12: 19th Conference on Computer and Communications Security*, pages 784–796, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[BPR+08] Dan Boneh, Periklis A. Papakonstantinou, Charles Rackoff, Yevgeniy Vahlis, and Brent Waters. On the impossibility of basing identity based encryption on trapdoor permutations. In *49th Annual Symposium on Foundations of Computer Science*, pages 283–292, Philadelphia, PA, USA, October 25–28, 2008. IEEE Computer Society Press.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. CRYPTO, 2017. (to appear).

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 523–552, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *8th IMA International Conference on Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363, Cirencester, UK, December 17–19, 2001. Springer, Heidelberg, Germany.

[DG17] Nico Döttling and Sanjam Garg. From selective ibe to full ibe and selective hibe. Manuscript, 2017.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GH09] Craig Gentry and Shai Halevi. Hierarchical identity based encryption with polynomially many levels. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 437–456. Springer, Heidelberg, Germany, March 15–17, 2009.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 405–422, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled RAM. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 210–229, Berkeley, CA, USA, October 17–20, 2015. IEEE Computer Society Press.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 449–458, Portland, OR, USA, June 14–17, 2015. ACM Press.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

[GS02] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany.

[HK09] Dennis Hofheinz and Eike Kiltz. The group of signed quadratic residues and applications. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 637–653, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[KR98] Hugo Krawczyk and Tal Rabin. Chameleon hashing and signatures. Cryptology ePrint Archive, Report 1998/010, 1998. http://eprint.iacr.org/1998/010.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 719–734, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 455–479, Zurich, Switzerland, February 9–11, 2010. Springer, Heidelberg, Germany.

[McC88] Kevin S. McCurley. A key distribution system equivalent to factoring. *Journal of Cryptology*, 1(2):95–105, 1988.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology – CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426, Santa Barbara, CA, USA, August 18–22, 1986. Springer, Heidelberg, Germany.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *21st Annual ACM Symposium on Theory of Computing*, pages 33–43, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 191–208, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[PRV12] Periklis A. Papakonstantinou, Charles W. Rackoff, and Yevgeniy Vahlis. How powerful are the DDH hard groups? Cryptology ePrint Archive, Report 2012/653, 2012. http://eprint.iacr.org/2012/653.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

[Shm85] Z. Shmuely. Composite diffie-hellman public-key generating systems are hard to break. Technical Report No. 356, Computer Science Department, Technion, Israel, 1985.

[SW08] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 560–578, Reykjavik, Iceland, July 7–11, 2008. Springer, Heidelberg, Germany.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication

Conference Paper · July 2007

DOI: 10.1007/978-3-540-73458-1_8 · Source: DBLP

The 12th Australasian Conference on Information Security and Privacy (ACISP '07).
(2–4 july 2007, Townsville, Queensland, Australia)
J. Pieprzyk Ed. Springer-Verlag, LNCS ????, pages ???–???.

An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication*

Julien Bringer[1], Hervé Chabanne[1], Malika Izabachène[2], David Pointcheval[2], Qiang Tang[2] and Sébastien Zimmer[2]

[1] Sagem Défense Sécurité
[2] Departement d'Informatique, École Normale Supérieure
45 Rue d'Ulm, 75230 Paris Cedex 05, France

Abstract This work deals with the security challenges in authentication protocols employing volatile biometric features, where the authentication is indeed a comparison between a fresh biometric template and that enrolled during the enrollment phase. We propose a security model for biometric-based authentication protocols by assuming that the biometric features to be public. Extra attention is paid to the privacy issues related to the sensitive relationship between a biometric feature and the relevant identity. Relying on the Goldwasser-Micali encryption scheme, we introduce a protocol for biometric-based authentication and prove its security in our security model.

Keywords. Authentication, biometrics, privacy.

1 Introduction

Security protocols generally rely on exact knowledge of some data, such as a cryptographic key, however there are particular applications where environment and human participation generate variability. In biometric-based cryptosystems, when a user identifies or authenticates himself using his biometrics, the biometric feature, which is captured by a sensor (e.g. a camera for iris biometrics), will rarely be the same twice. Thus, traditional cryptographic handling such as a hash value is not suitable in this case, since it is not error tolerant. As a result, the identification or authentication must be done in a special way, and moreover precaution is required to protect the sensitivity (or privacy) of biometrics.

We here consider a practical environment where a human user wants to authenticate himself to a database using his biometrics. A typical scenario is that some reference biometric data is stored inside a database, through which the server authenticates the user by checking whether or not a "fresh" biometric template sent by the sensor matches with the reference one. Our main focus is about biometrics such as iris [4], which can be extracted into binary strings. Therefore, an authentication leads to a comparison between two binary vectors. If the Hamming distance is adopted, then a comparison consists of computing the Hamming distance between the reference data and the fresh template and comparing this to a threshold.

To enforce privacy, we wish biometric data after their capture to be hidden in some way so that an adversary is unable to find out who is the real person that is trying to authenticate himself. Note that a live person is uniquely identified by

* Work partially supported by french ANR RNRT project BACH.

his biometrics and we want to hide the relationship between biometrics and the identity (used in an application). To achieve this goal, an application dependent identity is used and biometric matching is made over encrypted data. Moreover, to retrieve data to be compared with from the database, we introduce a new protocol to hide the index of record from the database.

1.1 Related Works

In [8] Juels and Wattenberg start the pioneering work by combining error correction codes with biometrics to construct fuzzy commitment schemes. Later on two important concepts about, i.e., secure sketch and fuzzy extractor, are widely studied. In [9], a number of secure sketch schemes have been proposed. In [6], Dodis *et al.* formalize the concept of fuzzy extractor, and propose to use for symmetric key generation from biometric features. In [2], Boyen *et al.* propose applications to remote biometric authentication using biometric information. Moreover, the work of Linnartz and Tuyls [10] investigates key extraction generated from continuous sources. In these schemes, biometric features are treated to be secret and used to derive general symmetric keys for traditional cryptographic systems.

There are a number of papers which deal with the secure comparison of two binary strings without using error correcting codes. In the protocol proposed by Atallah *et al.* [1], biometric features are measured as bit strings and subsequently masked and permuted during the authentication process. The comparison of two binary vectors modified following the same random transformation leads then to the knowledge of the Hamming distance. The main drawback of their protocol is that the client needs to store a number of secret values and update them during every authentication process, as the security relies mainly on these transformations.

Cryptographic protocols using homomorphic encryption may also allow us to compare directly encrypted data. For instance, Schoenmakers and Tuyls improve Paillier's public encryption protocol and propose to use it for biometric authentication protocols by employing multi-party computation techniques [12].

In summary, most of these protocols, except the work of [11] which uses biometry for Identity-Based Encryption, rely on the assumption that biometric features belonging to live users are private information. However, this assumption is not true in practice. As a user's biometric information, such as fingerprint, may be easily captured in daily life. In this paper, we assume that the biometric information is public, but the relationship between a user's identity and its biometric information is private.

1.2 Our contributions

In this paper we propose a general security model for biometric-based authentication. The model possesses a number of advantages over the existing ones: The first is that we lower the level of trust on the involved individual principals. The

second is that extra attention has been paid to the privacy issues related to the sensitive relationship between a biometric feature and the relevant identities. Specifically, this relationship is unknown to the database and the matcher.

We propose a new biometric authentication protocol which is proved secure in our security model. Our protocol follows a special procedure to query the database, which, as in the case of Private Information Retrieval (PIR) protocol [3], allows to retrieve an item without revealing which item is retrieved. The protocol heavily exploits the homomorphic property of Goldwasser-Micali public-key encryption scheme [7], its ability to treat plaintext bit after bit, and the security is based on its semantic security, namely the quadratic residuosity assumption.

1.3 Organization of this Work

The rest of the paper is organized as follows. In Section 2, we describe our security model for (remote) biometric-based authentication. In Section 3, we describe a new protocol for biometric authentication. In Section 4, we give the security analysis of the new protocol in the new security model. In Section 5, we conclude the paper.

2 A New Security Model

For a biometric-based remote authentication system, we assume the system mainly consists of two parts: the client part and the server part. At the client side, we distinguish the following two types of entities:

- A human being U_i, for any $i \geq 1$, who registers his reference biometric template b_i at the server side, and provides fresh biometric information in order to obtain any service from the authentication server.
- A sensor $\mathcal{S}$ which is capable of capturing the user's biometric and extracting it into a binary string, namely a fresh template.

In practice, the template extraction process may involve a number of components, nonetheless, here we assume that the sensor implements all these functionalities. Implicitly, we assume that the sensor can communicate with the server.

At the server side, we distinguish the following three types of entities:

- An authentication server, denoted $\mathcal{AS}$, which deals with the user's service requests and provides the requested service.
- A database $\mathcal{DB}$, which stores users' biometric templates.
- A matcher $\mathcal{M}$, which helps the server to make a decision related to a user's request of authentication.

Fig. 1 below illustrates this model.

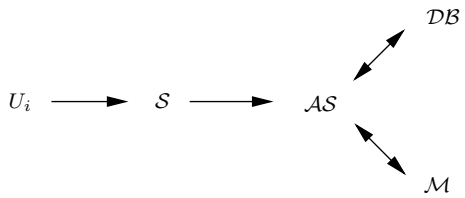


Figure1. Our model

Like most existing biometric-based systems (and many traditional cryptosystems), in our security model, a biometric-based authentication protocol consists of two phases: an enrollment phase and a verification phase.

1. In the enrollment phase, U_i registers its biometric template b_i at the database $\mathcal{DB}$ and its identity information ID_i at the server $\mathcal{AS}$.
2. In the verification phase, U_i issues an authentication request to the server $\mathcal{AS}$ through the sensor $\mathcal{S}$. The server $\mathcal{AS}$ retrieves U_i's biometric information from the database $\mathcal{DB}$ and makes its decision with the help of $\mathcal{M}$.

We assume that a "liveness link" is always available between the sensor $\mathcal{S}$ and the authentication server $\mathcal{AS}$ to ensure $\mathcal{AS}$ that the biometric it receives is from a present living person. The possible methods to achieve this liveness link are beyond the scope of this paper, but one can think about organizational measures or technical anti-spoofing countermeasures as those described in [5]. In addition, classical cryptographic challenge / response may also be used. This liveness link ensures that the server do not receive fake or replayed data. Since the sensor $\mathcal{S}$ is responsible for processing the biometric features, hence, it should be fully trusted and extensively protected in practice. Implicitly, the communications at the server side are also properly protected in the sense of authenticity. We further assume that all principals in the system will not collude and be honest-but-curious, which means they will not deviate from the protocol specification. In practice, certain management measures may be used to guarantee this assumption.

Let $\mathcal{H}$ be the distance function in the underlying metric space, for instance the Hamming space in our case. We regard soundness as a pre-requisite of any useful protocol. Formally, we have the following requirement.

Requirement 1 *The matcher $\mathcal{M}$ can faithfully compute the distance $\mathcal{H}(b_i, b_i')$, where b_i is the reference biometric template and b_i' is the fresh biometric template sent in the authentication request. Therefore, $\mathcal{M}$ can compare the distance to a given threshold value d and the server $\mathcal{AS}$ can make the right decision.*

Our main concern is the sensitive relationship between U_i's identity and its biometrics. We want to guarantee that any principal except for the sensor $\mathcal{S}$ cannot find any information about the relationship. Formally, we have the following requirement.

Requirement 2 *For any identity ID_{i_0}, two biometric templates b'_{i_0}, b'_{i_1}, where $i_0, i_1 \geq 1$ and b'_{i_0} is the biometric template related to ID_{i_0}, it is infeasible for any of $\mathcal{M}$, $\mathcal{DB}$, and $\mathcal{AS}$ to distinguish between (ID_{i_0}, b'_{i_0}) and (ID_{i_0}, b'_{i_1}).*

We further want to guarantee that the database $\mathcal{DB}$ gets no information about which user is authenticating himself to the server. Formally, we have the following requirement.

Requirement 3 *For any two users U_{i_0} and U_{i_1}, where $i_0, i_1 \geq 1$, if U_{i_β} where $\beta \in \{0,1\}$ makes an authentication attempt, then the database $\mathcal{DB}$ can only guess β with a negligible advantage. Suppose the database $\mathcal{DB}$ makes a guess β', the advantage is $|\Pr[\beta = \beta'] - \frac{1}{2}|$.*

3 A New Biometric-based Authentication Protocol

3.1 Review of the Goldwasser-Micali Scheme

The algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ of Goldwasser-Micali scheme [7] are defined as follows:

1. The key generation algorithm $\mathcal{K}$ takes a security parameter 1^ℓ as input, and generates two large prime numbers p and q, $n = pq$ and a non-residue x for which the Jacobi symbol is 1. The public key pk is (x, n), and the secret key sk is (p, q).
2. The encryption algorithm $\mathcal{E}$ takes a message $m \in \{0,1\}$ and the public key (x, n) as input, and outputs the ciphertext c, where $c = y^2 x^m \mod n$ and y is randomly chosen from $\mathbb{Z}_n^*$.
3. The decryption algorithm $\mathcal{D}$ takes a ciphertext c and the private key (p, q) as input, and outputs the message m, where $m = 0$ if c is a quadratic residue, $m = 1$ otherwise.

It is well-known (cf. [7]) that, if the quadratic residuosity problem is intractable, then the Goldwasser-Micali scheme is semantically secure. In other words an adversary $\mathcal{A}$ has only a negligible advantage in the following game.

$$
\begin{array}{l}
\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{E},\mathcal{A}} \\
\left|\begin{array}{rl}
(sk, pk) & \leftarrow \mathcal{K}(1^\ell) \\
(m_0, m_1) & \leftarrow \mathcal{A}(pk) \\
c & \leftarrow \mathcal{E}(m_\beta, pk),\ \beta \leftarrow \{0,1\} \\
\beta' & \leftarrow \mathcal{A}(m_0, m_1, c, pk)
\end{array}\right. \\
\text{return } \beta'
\end{array}
$$

At the end of this game, the attacker's advantage $\mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{E},\mathcal{A}}$ is defined to be

$$\mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{E},\mathcal{A}} = |Pr[\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{E},\mathcal{A}} = 1|\beta = 1] - Pr[\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{E},\mathcal{A}} = 1|\beta = 0]|.$$

Moreover the encryption protocol possesses a nice homomorphic property, for any $m, m' \in \{0,1\}$ the following equation holds.

$$\mathcal{D}(\mathcal{E}(m, pk) \times \mathcal{E}(m', pk), sk) = m \oplus m'$$

Note that the encryption algorithm encrypts one bit at a time, hence, in order to encrypt a binary string we need to encrypt every bit individually. We thus have the following property.

Lemma 1 ([7]) *Given any $M \geq 1$, the attacker's advantage in the following game is negligible based on the quadratic residuosity assumption.*

$\mathbf{Exp}^{P\text{-}IND\text{-}CPA}_{\mathcal{E},\mathcal{A}'}$

$$\begin{array}{|rl} (sk, pk) & \leftarrow \mathcal{K}(1^\ell) \\ ((m_{0,1}, \ldots, m_{0,M}), (m_{1,1}, \ldots, m_{1,M})) & \leftarrow \mathcal{A}'(pk) \\ c & \leftarrow (\mathcal{E}(m_{\beta,1}, pk), \ldots, \mathcal{E}(m_{\beta,M}, pk)),\ \beta \leftarrow \{0,1\} \\ \beta' & \leftarrow \mathcal{A}'((m_{0,1}, \ldots, m_{0,M}), (m_{1,1}, \ldots, m_{1,M}), c, pk) \\ \textit{return } \beta' & \end{array}$$

3.2 Enrollment Phase

In the protocol we treat U_i's biometric template b_i as a binary vector of the dimension M, i.e. $b_i = (b_{i,1}, b_{i,2}, \ldots, b_{i,M})$.

In the enrollment phase, U_i registers (b_i, i) at the database $\mathcal{DB}$, and (ID_i, i) at the authentication server $\mathcal{AS}$, where ID_i is U_i's pseudonym and i is the index of the record b_i in $\mathcal{DB}$. Let N denotes the total number of records in $\mathcal{DB}$.

The matcher $\mathcal{M}$ possesses a key pair (pk, sk) for the Goldwasser-Micali scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $pk = (x, n)$ and $sk = (p, q)$.

3.3 Verification Phase

If the user U_i wants to authenticate himself to the authentication server $\mathcal{AS}$, the procedure below is followed:

1. The sensor $\mathcal{S}$ captures the user's biometric data b'_i, and sends $\mathcal{E}(b'_i, pk)$ together with the user's identity ID_i to the authentication server $\mathcal{AS}$, where

   $$\mathcal{E}(b'_i, pk) = (\mathcal{E}(b'_{i,1}, pk), \mathcal{E}(b'_{i,2}, pk), \ldots, \mathcal{E}(b'_{i,M}, pk)).$$

   Note that a "liveness link" is available between $\mathcal{S}$ and $\mathcal{AS}$ to ensure that data coming from the sensor are indeed fresh and not artificial.
2. The server $\mathcal{AS}$ retrieves the index i using ID_i, and then sends $\mathcal{E}(t_j, pk)$ $(1 \leq j \leq N)$ to the database, where $t_j = 1$ if $j = i$, $t_j = 0$ otherwise.
3. For every $1 \leq k \leq M$, the database $\mathcal{DB}$ computes $\mathcal{E}(b_{i,k}, pk)$, where

   $$\mathcal{E}(b_{i,k}, pk) = \prod_{j=1}^{N} \mathcal{E}(t_j, pk)^{b_{j,k}} \mod n,$$

   Then it sends these $\mathcal{E}(b_{i,k}, pk)$ $(1 \leq k \leq M)$ to the authentication server $\mathcal{AS}$.

4. The authentication server $\mathcal{AS}$ computes ν_k $(1 \leq k \leq M)$, where

$$\begin{aligned} \nu_k &= \mathcal{E}(b'_{i,k}, pk)\mathcal{E}(b_{i,k}, pk) \mod n \\ &= \mathcal{E}(b'_{i,k} \oplus b_{i,k}, pk) \end{aligned}$$

   It then makes a random permutation among ν_k $(1 \leq k \leq M)$ and sends the permuted vector λ_k $(1 \leq k \leq M)$ to the matcher $\mathcal{M}$.
5. The matcher $\mathcal{M}$ decrypts the λ_k $(1 \leq k \leq M)$ to check if the Hamming weight of the corresponding plaintext vector is equal to or less than d, and sends the result to $\mathcal{AS}$.
6. The authentication server $\mathcal{AS}$ accepts or rejects the authentication request accordingly.

To sum up, $\mathcal{S}$ stores the public key pk, $\mathcal{AS}$ stores the public key pk and a table of relations (ID_i, i) for $i \in \{1, \ldots, N\}$, $\mathcal{DB}$ contains the enrolled biometric data $b_1, \ldots, b_N$, and $\mathcal{M}$ possesses the secret key sk, then the protocol runs following Fig. 2.

$\mathcal{S}$ Capture b'_i from U_i	$\xrightarrow{\mathcal{E}(b'_i,pk),ID_i}$	$\mathcal{AS}$
$\mathcal{AS}$ Choose $t_j = \delta_{i,j}$	$\xrightarrow{(\mathcal{E}(t_j,pk))_{1\leq j\leq N}}$	$\mathcal{DB}$
$\mathcal{AS}$	$\xleftarrow{(\mathcal{E}(b_{i,k},pk))_{1\leq k\leq M}}$	For $1 \leq k \leq M$, compute $\prod_{j=1}^{N} \mathcal{E}(t_j, pk)^{b_{j,k}} \mod n$ $= \mathcal{E}(b_{i,k}, pk)$
Compute $\nu_k = \mathcal{E}(b'_{i,k}, pk)\mathcal{E}(b_{i,k}, pk) \mod n$ $= \mathcal{E}(b'_{i,k} \oplus b_{i,k}, pk)$ Take a random permutation σ, compute $\lambda_k = \nu_{\sigma(k)}$	$\xrightarrow{\lambda_1,\ldots,\lambda_M}$	$\mathcal{M}$ Check the weight of $(\mathcal{D}(\lambda_1, sk), \ldots, \mathcal{D}(\lambda_M, sk))$
$\mathcal{AS}$	$\xleftarrow{\text{OK / NOK}}$	$\mathcal{DB}$

Figure2. The Authentication protocol

It is easy to verify that the sensor $\mathcal{S}$ performs at most $2M$ modular multiplications, the server performs $2N$ modular multiplications in step 2 (which can be pre-computed) and M modular multiplications in step 4. The database needs to perform $\frac{MN}{2}$ modular multiplications in step 3, if we assume that 0 and 1 are equally distributed in the set $\{b_{j,k}\}_{1\leq j\leq N, 1\leq k\leq M}$. The matcher performs M modular exponentiations to check quadratic residuosity modulo p. And the overall communication complexity is linear on the number N of records in the database.

4 Security Analysis of the Protocol

The introduction of the matcher $\mathcal{M}$, which holds the decryption key, effectively limits the access to users' biometric information. The matcher $\mathcal{M}$ can only obtain the Hamming distance between two measurements of any user's biometrics, which actually can be thought of being public information. The server does not store any biometric information, hence, compromise of the server leaks no information to an outside attacker. Moreover, biometrics are almost always handled in an encrypted form.

Indeed the biometric templates are stored in plaintext in the database $\mathcal{DB}$, however, without any relevant identity information. In case that the database is compromised, no sensitive relationship information would be leaked, though we consider encrypting the biometric templates in the database is an interesting future research topic.

In the next section we show that the protocol satisfies the requirements described in Section 2.

4.1 Fulfillment of our Requirements

In step 4 of the protocol, we show that $\nu_k = \mathcal{E}(b'_{i,k} \oplus b_{i,k}, pk)$ for $1 \leq k \leq M$. Obviously, the Hamming distance between b_i and b'_i, $\mathcal{H}(b_i, b'_i)$, is equal to the Hamming weight of the plaintext vector corresponding to $(\nu_1, \ldots, \nu_M)$ and $(\lambda_1, \ldots, \lambda_M)$. Hence, it is straightforward to verify that **Requirement 1** is fulfilled.

We next show that the authentication protocol satisfies **Requirement 2** under the quadratic residuosity assumption.

Theorem 1 *For any identity ID_{i_0} and two biometric templates b'_{i_0}, b'_{i_1}, where $i_0, i_1 \geq 1$ and b'_{i_0} is the biometric template related to ID_{i_0}, any of $\mathcal{M}$, $\mathcal{DB}$, and $\mathcal{AS}$ can only distinguish between (ID_{i_0}, b'_{i_0}) and (ID_{i_0}, b'_{i_1}) with a negligible advantage.*

Proof. It is clear that the matcher $\mathcal{M}$ and the database $\mathcal{DB}$ have advantage 0 in distinguishing between (ID_{i_0}, b'_{i_0}) and (ID_{i_0}, b'_{i_1}), because they have no access to any information about users' identities.

As to the server $\mathcal{AS}$, the proof follows. From (ID_{i_0}, b'_{i_β}) with $\beta \in \{0, 1\}$, if the database $\mathcal{AS}$ can guess β with a non-negligible advantage δ, then we construct an attacker $\mathcal{A}$ for the Goldwasser-Micali scheme (as defined in Lemma 1) which has the advantage δ. The attacker simulates the protocol executions for the server $\mathcal{AS}$.

Suppose $\mathcal{A}$ receives pk from the challenger and gets a challenge $c_d = \mathcal{E}(m_{i_d}, pk)$ for $m_{i_0} \neq m_{i_1}$, where d is a random bit chosen by the challenger. $\mathcal{A}$ simulates the protocol executions by assuming that the matcher $\mathcal{M}$ and the database $\mathcal{DB}$ take pk as the public key. Then $\mathcal{A}$ registers m_{i_0} and m_{i_1} in the database. Note that it is straightforward to verify that the protocol execution for $\mathcal{AS}$ can be

faithfully simulated by $\mathcal{A}$, and the knowledge of private key sk is not needed. If the server $\mathcal{AS}$ outputs a guess β', then $\mathcal{A}$ outputs the guess bit $d' = \beta'$ for d. As $\mathcal{A}$ wins if $\mathcal{AS}$ wins, the theorem now follows from Lemma 1. □

Now we prove that the authentication protocol also satisfies **Requirement 3** under the quadratic residuosity assumption.

Theorem 2 *For any two users U_{i_0} and U_{i_1}, where $i_0, i_1 \geq 1$, if U_{i_β} where $\beta \in \{0,1\}$ makes an authentication attempt, then the database $\mathcal{DB}$ can only guess β with a negligible advantage.*

Proof. If the database $\mathcal{DB}$ can guess β with a non-negligible advantage δ, then we construct an attacker $\mathcal{A}$ for the Goldwasser-Micali scheme which has the advantage δ.

Suppose $\mathcal{A}$ receives pk from the challenger and gets a challenge $c_d = \mathcal{E}(m_d, pk)$ for $m_0 = 0, m_1 = 1$, where d is a random bit chosen by the challenger. In addition, $\mathcal{DB}$ takes pk as the matcher's public key. For any $i_0, i_1 \geq 1$ and $i_0 \neq i_1$, $\mathcal{A}$ issues a query with $\mathcal{E}(t_j, pk)$ $(1 \leq j \leq N)$, where $\mathcal{E}(t_{i_1}, pk) = c_d$, $\mathcal{E}(t_{i_0}, pk) = y^2 x c_d$ where y is randomly chosen from $\mathbb{Z}_n^*$, and $t_j = 0$ for all $1 \leq j \leq N, j \neq i_0, j \neq i_1$. If the database $\mathcal{DB}$ outputs a guess β', then $\mathcal{A}$ outputs the guess bit $d' = \beta'$ for d. And it is straightforward to verify that $\mathcal{A}$ wins if $\mathcal{DB}$ wins. □

4.2 Advantages of the protocol

To emphasize the interest of our protocol, we further compare it with one recent protocol of Atallah *et al.* [1] which also allows the comparison between two binary biometric templates.

In the protocol of Atallah *et al.* [1] two entities are involved: a server which stores some information about the reference data b and a client (with a biometric sensor) which sends other information derived from the measured data b'. In the initialization phase, the client stores a random permutation Π_1 of $\{0,1\}^n$ and three random boolean vectors s_1, s_2, r_1. The client then sends $s_1 \oplus \Pi_1(b_1 \oplus r_1), \mathsf{H}(s_1), \mathsf{H}(s_1, \mathsf{H}(s_2))$ to the server for backup, where H is a hash function and b_1 is the user's biometric data. When measuring a new features vector b_2, the client sends $s_1, \Pi_1(b_2 \oplus r_1)$ to the server which could then verify the value of $\mathsf{H}(s_1)$ and compute the Hamming distance of b_1, b_2 to check if it is in an acceptable range. Thereafter, the remaining vectors are used to renew all the information stored at the client and the server sides for a future authentication.

The main drawback of this protocol is that the client needs to store secret values. Once these values are compromised, the attacker would be able to compute a user's biometric template easily by passively eavesdropping on the communication channel. It is also possible to show that an active attacker could impersonate the client to the server. Finally, it is also clear that the user's privacy is not ensured against the server. Therefore, it makes sense for us to explore new protocols that avoid these drawbacks.

Hence, the most important points that make our protocol more appropriate for biometrics authentication protocols are the following. Firstly, no secret infor-

mation storage is required at the client side. Secondly, the protocol guarantees the privacy of the relationship between the user's identity and its biometric data, and the privacy of the user's biometric information.

5 Conclusion

In this paper, we considered a biometric authentication protocol where confidentiality is required for biometric data solely for privacy reasons. We captured these notions into a security model and introduced a protocol which is proved secure in this security model. It remains an interesting issue to improve its performance. For a better acceptability, we also want to look at an extension of this work where biometric data inside the database are also encrypted.

Acknowledgments

We would like to thank Michel Abdalla for the fruitful discussions.

References

1. M. J. Atallah, K. B. Frikken, M.l T. Goodrich, and R. Tamassia. Secure biometric authentication for weak computational devices. In A. S. Patrick and M. Yung, editors, *Financial Cryptography*, volume 3570 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2005.
2. X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith. Secure remote authentication using biometric data. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2005.
3. B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
4. J. Daugman. How iris recognition works. In *ICIP (1)*, pages 33–36, 2002.
5. J. Daugman. Iris recognition and anti-spoofing countermeasures. In 7*-th International Biometrics Conference*, 2004.
6. Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer, 2004.
7. S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, 5-7 May 1982, San Francisco, California, USA*, pages 365–377. ACM, 1982.
8. A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *ACM Conference on Computer and Communications Security*, pages 28–36, 1999.
9. Q. Li and E. Chang. Robust, short and sensitive authentication tags using secure sketch. In *MM&Sec '06: Proceeding of the 8th workshop on Multimedia and security*, pages 56–61. ACM Press, 2006.
10. Jean-Paul M. G. Linnartz and Pim Tuyls. New shielding functions to enhance privacy and prevent misuse of biometric templates. In Josef Kittler and Mark S. Nixon, editors, *AVBPA*, volume 2688 of *Lecture Notes in Computer Science*, pages 393–402. Springer, 2003.
11. A. Sahai and B. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.
12. B. Schoenmakers and P. Tuyls. Efficient binary conversion for Paillier encrypted values. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 522–537. Springer, 2006.

IBM SPSS Statistics
Version 25

Linux Installation Instructions (Concurrent License)

Contents

Installation instructions

The following instructions are for installing IBM® SPSS® Statistics version 25 using the license type concurrent license. This document is for users who are installing on their desktop computers.

System requirements

To view system requirements, go to http://publib.boulder.ibm.com/infocenter/prodguid/v1r0/clarity/index.jsp.

Installing

Important: To install, you must run the installation program as *root*.

Running multiple versions and upgrading from a previous release

You do not need to uninstall an old version of IBM SPSS Statistics before installing the new version. Multiple versions can be installed and run on the same machine. However, do not install the new version in the same directory in which a previous version is installed.

Note for IBM SPSS Statistics Developer

If you are installing IBM SPSS Statistics Developer, you can run the product standalone. If you accept the default option to install IBM SPSS Statistics - Essentials for Python, then you have the tools to develop with Python. You can also install IBM SPSS Statistics - Essentials for R to develop with R. It is available from the SPSS Community at https://developer.ibm.com/predictiveanalytics/predictive-extensions/.

Installing from a downloaded file

1. Start a terminal application.
2. Change to the directory where you downloaded the file.
3. Extract the contents of the file.
4. Change to the directory where you extracted the files.
5. Make sure the permissions for *setup.bin* are set to execute.
6. At the command prompt, type:
   ```
   ./setup.bin
   ```
7. Follow the instructions that appear in the installation program. See “Notes for installation” on page 2 for any special instructions.

Installing from the DVD/CD

1. Insert the DVD/CD into your DVD/CD drive.
2. Start a terminal application.
3. Change the directory to the DVD/CD mount point.
4. At the command prompt, type:
   ```
   ./Linux/setup.bin
   ```
   Note: The previous command will work only if you mounted the DVD/CD with the option to execute binary files. As an alternative, you can create a copy of the DVD/CD locally. Go to the *Linux* directory in the local copy and make sure the permissions for *setup.bin* are set to execute and then run *setup.bin*.
5. Follow the instructions that appear in the installation program. See “Notes for installation” on page 2 for any special instructions.

Notes for installation

This section contains special instructions for this installation.

Installer language. The first panel of the installer prompts for an installer language. By default, the language that matches your locale is selected. If you would like to display the installer in another language, select the language. Click **OK** when you are ready to proceed.

IBM SPSS Statistics - Essentials for Python. You are prompted to install IBM SPSS Statistics - Essentials for Python. Essentials for Python provides you with the tools to develop custom Python applications for use with IBM SPSS Statistics, and to run extension commands that are implemented in the Python language. It includes Python versions 2.7 and 3.4, the IBM SPSS Statistics - Integration Plug-in for Python, and a set of Python extension commands that provide capabilities beyond what is available with built-in SPSS Statistics procedures. For more information, see Integration Plug-in for Python in the Help system. To accept the default option to install IBM SPSS Statistics - Essentials for Python, you must accept the associated license agreement.

Licensing your product

You must run the License Authorization Wizard to license your product.

Using the license authorization wizard

1. To launch the License Authorization Wizard, run the *licensewizard* file in the *bin* subdirectory of the installation directory. Like the installer file, run this as *root*.
2. Select **Concurrent user license**. When prompted, enter the license manager server name or IP address. This is the IP address or the name of the server on which the network license manager is running. If you have multiple addresses or names, separate them with a tilde (for example, `server1~server2~server3`). Contact your administrator if you do not have this information.

   Note: Depending on your environment, you may need verify that TCP port 7 is open. The License Authorization Wizard needs to contact the license manager server one time on port 7 to verify it exists.

Running the application remotely

You must run IBM SPSS Statistics on the physical machine on which it is installed. You cannot use the `DISPLAY` environment variable to run IBM SPSS Statistics from a remote machine.

Starting IBM SPSS Statistics

1. Browse to the *bin* subdirectory in the installation directory.
2. Run the *stats* file.

Checking out/in a commuter license

Network licenses normally require that you are connected to the network to run IBM SPSS Statistics. If your administrator enabled commuter licenses, you can check out a commuter license to use the network license when you are not connected to the network. For example, you may want to run IBM SPSS Statistics on the train when you don't have a network connection. Before disconnecting from your network and catching the train, you could check out a commuter license for a limited amount of time. You will need to reconnect to the network and check the license back in before the time expires. Otherwise, IBM SPSS Statistics will stop working.

Network licenses are enabled and configured by your administrator. If you would like to use this feature and can't, check with your administrator.

Important: Even if you are able to run IBM SPSS Statistics because you are reconnected to the network, be sure to check the license back in. Doing so will allow other users to take advantage of the commuter license.

Check out a license

1. Run the *licensecommute* script in the *bin* subdirectory of the product installation directory.
2. Select the license that you want to check out.
3. In the Duration box, enter the number of days for which you want to check out the license. There is a limit that your administrator configures.
4. Click **Check Out**.

The commuter license will expire after the number of days specified by **Duration**. You can also manually check the license back in at any time.

You may receive a message in the following format:

```
Error while checkout with error code: <code>
```

Common codes are as follows.

Code	Meaning
77	All available licenses have been checked out.
1402	Attempt to check out license that has been reserved for another user.

Check in a license

1. Run the *licensecommute* script in the *bin* subdirectory of the product installation directory.
2. Select the license that you want to check in. License(s) that you checked out are indicated by a check mark.
3. Click **Check In**.

Applying fix packs

To ensure problem-free operation, keep your product at the latest fix pack level. Complete all of the necessary pre-installation and post-installation tasks as described in the fix pack instructions.

Uninstalling

1. Start a terminal program.
2. Change the directory to *Uninstall_IBM SPSS Statistics 25* in the IBM SPSS Statistics installation directory.
3. At the command prompt, type:

   ./Uninstall_IBM SPSS Statistics_25

   Important: You must have permissions to remove the installation directory, or the uninstallation process will fail.
4. Follow the instructions that appear in the uninstallation program.
5. Delete the folder *~/.IBM/SPSS/Statistics/25/Eclipse*. In a terminal application, enter the following command:

   ```
   rm -fr ~/.IBM/SPSS/Statistics/25/Eclipse
   ```

Printed in USA

A chameleon encryption scheme resistant to known-plaintext attack

Article · January 2010

DOI: 10.1145/1866870.1866876

3 authors:

Ee-Chien Chang
National University of Singapore
131 PUBLICATIONS **2,193** CITATIONS
SEE PROFILE

Chengfang Fang
Huawei Technologies
16 PUBLICATIONS **99** CITATIONS
SEE PROFILE

Jia Xu
NUS-SingTel Cyber Security R&D Lab
20 PUBLICATIONS **375** CITATIONS
SEE PROFILE

A Chameleon Encryption Scheme Resistant to Known-Plaintext Attack

Ee-Chien Chang Chengfang Fang Jia Xu
Department of Computer Science
National University of Singapore
Republic of Singapore
{changec,c.fang,xujia}@comp.nus.edu.sg

ABSTRACT

From a ciphertext and a secret key assigned to a user, the decryption of a *Chameleon encryption* scheme produces a message which is the plaintext embedded with a watermark associated to the user. Most existing constructions of Chameleon encryption scheme are LUT (lookup table)-based, where a secret LUT plays the role of the master key and each user has a noisy version of the secret LUT. LUT-based methods have the limitation that the secrecy of the master key, under known-plaintext attack (KPA), relies on the difficulty in solving large linear system. In other words, with some knowledge of the plaintext, a dishonest user is able to derive the LUT, or an approximation of the LUT by solving a linear system. Resistance to such attack is crucial in the context of multimedia encryption since multimedia objects inherently contain high redundancies. Furthermore, for efficiency in decryption, the underlying linear system is likely to be sparse or not overly large, and hence can be solved using reasonable computing resource. In our experiment, a desktop PC is able to find a LUT (with 2^{16} entries) within 2 hours. We propose a scheme that is resistant to KPA. The core of the scheme is a MUTABLE-PRNG (Pseudo Random Number Generator) whereby different but similar sequences are generated from related seeds. We generate such sequence from multiple pseudo random sequences based on majority-vote, and enhance its performance using error-correcting code. The proposed scheme is very simple and it is easy to show that it is resistant to KPA under reasonable cryptographic assumptions. However, it is not clear how much information on the original plaintext is leaked from the watermarked copies. We analyze the scheme and quantify the information loss using average conditional entropy.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

DRM'10, October 4, 2010, Chicago, Illinois, USA.

General Terms

Algorithm, Security

Keywords

Client-side watermark embedding, Chameleon Encryption, Known-Plaintext Attack, MUTABLE-PRNG

1. INTRODUCTION

With the boom of Internet, digital right protection becomes more and more crucial and challenging. Apart from legal penalties, many techniques are devised to resolve or mitigate the threat of pirate. Watermarking is an important technique among them. In a typical watermarking application, the content distributor first embeds watermarks into different copies of the multimedia data on the server side, and then distributes the watermarked copies to different subscribed users. Some scenarios require client-side watermarking, whereby the watermark is only embedded on the client side. One way to achieve secure client-side embedding is through trusted hardware [20, 21]. Each subscribed user will receive a tamper-proof physical decoder which decrypts and embeds watermark into the streamed data on the fly. However, securing hardware is costly. An alternative is Chameleon encryption. This framework consists of two phases: in the key setup phase, through a secure point-to-point channel, each user u receives a personalized decryption key $\mathcal{K}_u$ and the content distributor receives a master encryption key $\mathcal{K}$, from a trusted key generator. In the service phase, only an insecure broadcast channel is required. All data will be encrypted under the encryption key $\mathcal{K}$ by the distributor before being broadcasted to all subscribed users. Each subscribed user u can decrypt the received data using the assigned key $\mathcal{K}_u$. Instead of getting the original data, the user obtains a watermarked copy that is different from but perceptually similar to the original.

Most existing Chameleon encryption schemes [4, 6, 2] are lookup table(LUT)-based: A secret LUT is used during encryption, whereas each user is assigned a distinct noisy version of LUT for decryption. There are two main parameters, L and S, where L denotes the number of entries in LUT, and S is the number of LUT entries selected to encrypt/decrypt a single character. The LUT-based encryption has the limitation that the secrecy of the LUT relies on the difficulty in solving a large linear system under known-plaintext attack. Given plaintext-ciphertext pairs, an attacker is able to derive the master key, by solving a linear system. It is easy to

extend the attack to a scenario where the attacker has a decryption key and only partial knowledge of the plaintext. In this scenario, by finding the solution to a linear least square system, the attacker can obtain an "approximate" master key that is significantly close to the original, and closer than any user's decryption key. Such attacks need to be addressed since multimedia data inherently contain high redundancies. Although compression may help to reduce redundancies, the watermark embedding typically has to be performed before the lossless compression. For instance, for JPEG, the watermark can be embedded in the DCT domain, but not after the lossless compression. Using larger LUT would force the attacker to solve a larger linear system and could deter attackers with limited resources. However, LUT is part of the key and thus it is desirable to keep it small. Furthermore, the parameter S is desired to be small for efficiency in encryption/decryption. This leads to a sparse linear system that can be handled using reasonable computing resource. In our experiment, for LUT with size $L = 2^{16}$, where each entry is a non-negative integer smaller than 2^{16} and $S = 80$, a "good" approximation of LUT can be found using a desktop PC within 2 hours.

We propose a new Chameleon encryption scheme which is simple but secure against known-plaintext attack. The scheme is a one-time pad cipher equipped with a MUTABLE-PRNG F: Given an input $\mathcal{S}$, one can find another short input $\mathcal{S}_u$, such that the Hamming distance between $\mathsf{F}(\mathcal{S})$ and $\mathsf{F}(\mathcal{S}_u)$ is small. Let $\mathcal{S}$ and $\mathcal{S}_u$ be the encryption key and decryption key respectively. One-time pad cipher is adopted to mask (unmask, respectively) the plaintext (ciphertext, respectively) with the generated pseudorandom bit-stream $\mathsf{F}(\mathcal{S})$ ($\mathsf{F}(\mathcal{S}_u)$, respectively). As a result, $\mathsf{F}(\mathcal{S}) \oplus \mathsf{F}(\mathcal{S}_u)$ is embedded into the decrypted data as watermark during the decryption, where $\oplus$ is the XOR operator.

To generate the sequence $\mathsf{F}(\mathcal{S})$, we use majority vote on multiple pseudorandom sequences. Specifically, $\mathcal{S}$ is a set of n seeds from which n pseudorandom sequences are generated. For each location i, the i-th bit of $\mathsf{F}(\mathcal{S})$ is set to the majority bit among all i-th bits in the n pseudorandom sequences. Similarly, we can generate the sequence $\mathsf{F}(\mathcal{S}_u)$ from $\mathcal{S}_u$, where $\mathcal{S}_u$ is a subset of $\mathcal{S}$. The sequence $\mathsf{F}(\mathcal{S}_u)$ generated from $\mathcal{S}_u$ correlates with, but is not identical to, the sequence $\mathsf{F}(\mathcal{S})$ generated from $\mathcal{S}$. To increase the correlations, we employ error correcting code (ECC).

The proposed scheme is very simple, and it is easy to show that the master key is secure under KPA, with the assumptions that each pseudorandom sequence is generated by some cryptographically secure PRNG. However, the same as LUT-based methods, our scheme suffers from collusion attack, in the sense that a few collusive users can easily derive the master key or a good approximation of the master key. In this paper, we do not address collusion attack. It is interesting to further investigate whether cryptographic techniques can be incorporated to "hide" the computation of majority and thus prevent collusion.

Our scheme reveals extra information of the original data besides the watermarked copy, due to information provided by the ECC and the internal states in the computation of majority bits. We analyze and quantify such information loss using the notion of average conditional entropy.

Contribution.

1. While it is already known that LUT-based Chameleon encryption schemes are vulnerable under KPA, previous Chameleon encryption schemes rely on large LUT to deter attack. We argue that for multimedia data, it is crucial to be secure under KPA and point out that the LUT can be easily found since the underlying linear system is likely to be sparse. Our experimental studies show that a rather large LUT (with $L = 2^{16}$ entries and $S = 80$) can be found using a desktop PC within 2 hours.
2. We propose a new Chameleon encryption scheme based on a specially designed MUTABLE-PRNG (Section 5). This MUTABLE-PRNG is in turn constructed using majority voting and a cryptographically secure PRNG. We show that this scheme is KPA-secure (Theorem 1) and give an upper bound on the entropy loss due to ECC and the internal states in the computation of majority voting.

Organization.

The rest of this paper is organized as follows: Section 2 discusses the related work and Section 3 defines the problem framework and security concerns. The known-plaintext attack on previous Chameleon encryption schemes is given in Section 4. We present the new Chameleon encryption scheme in Section 5 and analyze its security in Section 6. Section 7 presents a variant version of our proposed scheme and Section 8 closes this paper.

2. RELATED WORKS

Client side watermark embedding allows the distributor to distribute a unique copy of encrypted data, and the personalized watermark will be added into the data on the client side in a secure way. The hardware solution requires each legitimate user to have a dedicated secure hardware to embed the watermark, thus incurs a huge deployment cost. A secure software solution is much more preferable. Anderson *et al.* [4] proposed Chameleon Encryption scheme which binds decryption and watermark embedding together so that adversary cannot separate them. Adelsbach *et al.* [2] and Celik *et al.* [6] improved Chameleon Encryption. Adelsbach *et al.* briefly summarized previous works on joint fingerprinting and decryption [17, 13, 15]. Lemma *et al.* [14] proposed a similar method such that the content owner distributes personalized help data to each user for every decryption.

3. DEFINITIONS AND NOTATIONS

There are two different roles involved: a content distributor $\mathcal{D}$ and a set $\mathcal{U}$ of users. At the very beginning, $\mathcal{D}$ generates[1] a master encryption key $\mathcal{K}$ for himself/herself, and a personalized decryption key $\mathcal{K}_u$ for each user $u \in \mathcal{U}$. $\mathcal{D}$ delivers the decryption key $\mathcal{K}_u$ to user $u \in \mathcal{U}$ securely. After this setup, $\mathcal{D}$ can distribute his/her data or messages (e.g. digital movies, music, TV service) to all users (e.g. those who have paid $\mathcal{D}$ for such multimedia data) in $\mathcal{U}$ in this way:

1. $\mathcal{D}$ encrypts the message x using the encryption key $\mathcal{K}$:

[1]For simplicity, we assume the content distributor $\mathcal{D}$ is also the key generator.

$c \leftarrow \mathsf{Enc}(x; \mathcal{K})$, and distributes c to all users in $\mathcal{U}$ via (insecure) broadcast.

2. Each user $u \in \mathcal{U}$ decrypts c using the decryption key $\mathcal{K}_u$: $\widetilde{x}_u \leftarrow \mathsf{Dec}(c; \mathcal{K}_u)$.

Each decrypted message $\widetilde{x}_u$ is a (distinct) watermarked copy of the original message x. The watermark is embedded into $\widetilde{x}_u$ just in the same process of decryption on the client-side. The above model is a simplified "Joint Fingerprinting and Decryption" framework [2].

Security.

In this paper, the adversary is some user $u \in \mathcal{U}$ with a decryption key $\mathcal{K}_u$. We are concerned about the following security property:

1. Secrecy of master encryption key: A user $u \in \mathcal{U}$ wants to find a good approximation $\widetilde{\mathcal{K}}$ of the master encryption key $\mathcal{K}$, which is much closer to $\mathcal{K}$ than the decryption key $\mathcal{K}_u$. That is, $\mathsf{Dist}(\widetilde{\mathcal{K}}, \mathcal{K}) \leq \tau \cdot \mathsf{Dist}(\mathcal{K}_u, \mathcal{K})$ for some threshold[2] $\tau \in [0, 1)$, where $\mathsf{Dist}(\cdot, \cdot)$ is some appropriate distance function. In Section 4.2.2, we choose L^2-Norm as the distance function. We consider two forms of known-plaintext attack. In the first form, the adversary has a decryption key and some plaintext-ciphertext pairs. We investigate whether the adversary can find the master encryption key with such information. In the second form, the adversary has a decryption key, some ciphertexts and knows the distribution of their corresponding plaintexts.
2. Robustness of watermark: For any watermarking scheme, the watermarked message inevitably reveals some information about the original. In joint decryption and watermarking framework, the combination of ciphertext C and user decryption key $\mathcal{K}_u$ may potentially reveal more information. We require the difference between the two, i.e. the extra information revealed by $(C, \mathcal{K}_u)$ to be small. However, it is very difficult to formulize this notion in a rigorous way. In our paper, as an approximation, we require $\mathbf{H}(X|X_u) - \mathbf{H}(X|C, \mathcal{K}_u)$ to be small, where $\mathbf{H}(\cdot|\cdot)$ denotes the *average conditional entropy* [19], the plaintext X is sampled from the plaintext domain with uniform distribution, C is the ciphertext of X, $\mathcal{K}_u$ is the decryption key for user $u \in \mathcal{U}$, and X_u is the decrypted message for user u.

4. CHAMELEON ENCRYPTION SCHEMES UNDER KNOWN-PLAINTEXT ATTACK

We first describe some constructions of Chameleon encryption [4, 6, 2], and then show that these constructions are vulnerable under known-plaintext attack.

4.1 Existing Constructions

4.1.1 Chameleon Encryption.

Anderson *et al.* [4] proposed the first Chameleon encryption scheme. The scheme has a parameter L, which represents the size of lookup table (LUT).

1. Encryption key is a $L \times 1$ table $\mathbf{E}$, and each table entry $\mathbf{E}[i]$, $0 \leq i \leq L-1$, stores a 64-bit bit string.
2. Decryption key for user u is a table $\mathbf{D}_u$ of the same size, and for $0 \leq i \leq L-1$, $\mathbf{D}_u[i] = \mathbf{E}[i] \oplus w_{u,i}$, where $w_{u,i}$ is a 64-bit personalized watermark with very small hamming weight and $\oplus$ is the XOR operator. Watermark $(w_{u,1}, w_{u,2}, \ldots, w_{u,L})$ identifies user u.
3. Encryption of $(x_0, x_1, \ldots, x_{m-1})$, where $x_i \in \{0,1\}^{64}$, $0 \leq i \leq m-1$:
   (a) Choose a random seed s, and from s generate a pseudorandom sequence $t_1, t_2, \ldots, t_{4m}$, where $t_j \in [0, L-1], 1 \leq j \leq 4m$.
   (b) For each $0 \leq i \leq m-1$, $c_i \leftarrow x_i \oplus \mathbf{E}[t_{4i}] \oplus \mathbf{E}[t_{4i-1}] \oplus \mathbf{E}[t_{4i-2}] \oplus \mathbf{E}[t_{4i-3}]$.
   (c) Output $(s, c_0, c_1, \ldots, c_{m-1})$.
4. Decryption of $(s, c_0, c_1, \ldots, c_{m-1})$ using decryption key $\mathbf{D}_u$ by user u
   (a) From s generate a pseudorandom sequence $t_1, \ldots, t_{4m}$, where $t_j \in [0, L-1], 1 \leq j \leq 4m$.
   (b) For each $0 \leq i \leq m-1$,

$$\begin{aligned}\widetilde{x}_i &\leftarrow c_i \oplus \mathbf{D}_u[t_{4i}] \oplus \mathbf{D}_u[t_{4i-1}] \oplus \mathbf{D}_u[t_{4i-2}] \oplus \mathbf{D}_u[t_{4i-3}] \\ &= x_i \oplus (\mathbf{E}[t_{4i}] \oplus \mathbf{D}_u[t_{4i}]) \oplus (\mathbf{E}[t_{4i-1}] \oplus \mathbf{D}_u[t_{4i-1}]) \oplus \\ &\quad (\mathbf{E}[t_{4i-2}] \oplus \mathbf{D}_u[t_{4i-2}]) \oplus (\mathbf{E}[t_{4i-3}] \oplus \mathbf{D}_u[t_{4i-3}]) \\ &= x_i \oplus (w_{u,4i} \oplus w_{u,4i-1} \oplus w_{u,4i-2} \oplus w_{u,4i-3}) .\end{aligned}$$

   (c) Output $(\widetilde{x}_0, \widetilde{x}_1, \ldots \widetilde{x}_{m-1})$.

4.1.2 Chameleon encryption with algebraic addition and large S.

Celik *et al.* [6] gave a variant version of Chameleon encryption, which is similar to the first Chameleon encryption scheme, except that it uses usual algebraic addition and subtraction to replace XOR, and real numbers to replace bit strings. Their scheme has two system parameters, L and S, where L is the size of LUT table and S is the number of LUT entries selected to encrypt/decrypt a single character. The scheme can be described as follows.

1. Encryption key is a $L \times 1$ table $\mathbf{E}$, and each table entry $\mathbf{E}[i]$ is independently and identically sampled from a fixed probability distribution (e.g. Gaussian distribution).
2. Decryption key for user u is a table $\mathbf{D}_u$ of the same size, and for $0 \leq i \leq L-1$,

$$\mathbf{D}_u[i] = -\mathbf{E}[i] + \mathbf{W}_u[i],$$

   where $\mathbf{W}_u$ is a $L \times 1$ personalized watermark LUT for user u and the entries of $\mathbf{W}_u$ are independently and identically sampled from some fixed distribution (e.g. Gaussian distribution).
3. Encryption of $(x_0, x_1, x_2, \ldots, x_{m-1})$, where $x_i \in \mathbb{R}$, $0 \leq i \leq m-1$:
   (a) Choose a random seed s, and from s generate a pseudorandom sequence $\{t_{i,j} \in [0, L-1] : 0 \leq i \leq m-1, 0 \leq j \leq S-1\}$.
   (b) For each $0 \leq i \leq m-1$,

$$c_i \leftarrow x_i + \sum_{j=0}^{S-1} \mathbf{E}[t_{i,j}].$$

4. Decryption of $(s, c_0, c_1, \ldots, c_{m-1})$ using decryption key $\mathbf{D}_u$ by user u

[2]The value of threshold τ is determined by applications.

(a) From s generate a pseudorandom sequence $\{t_{i,j} \in [0, L-1] : 0 \le i \le m-1, 0 \le j \le S-1\}$.

(b) For each $0 \le i \le m-1$,

$$\widetilde{x}_i = c_i + \sum_{j=0}^{S-1} \mathbf{D}_u[t_{i,j}] = x_i + \sum_{j=0}^{S-1} \mathbf{W}_u[t_{i,j}].$$

(c) Output $(\widetilde{x}_0, \widetilde{x}_1, \ldots \widetilde{x}_{m-1})$.

Celik *et al.* [6] remarked that larger S complicates cryptanalysis under known-plaintext attack.

4.1.3 Fingercasting.

Adelsbach *et al.* [2] gave a variant of Chameleon encryption with proof of security. Their scheme replaces XOR operation with modular addition and subtraction with a large prime modulus.

4.2 Vulnerabilities in Existing Constructions

All of the above three constructions are not KPA-secure: given several pairs of plaintext-ciphertext, a malicious user who has a decryption key, can find the long term encryption key or a good approximation of it. Section 4.2.1 studies the case that the adversary has full knowledge of the plaintexts corresponding to some given ciphertexts, and Section 4.2.2 studies the case that the adversary only has partial information of the plaintexts corresponding to some given ciphertexts.

4.2.1 KPA with Full Knowledge of Plaintext.

As mentioned by [6], the LUT based encryption can be represented in matrix form

$$\vec{c} = \vec{x} + \mathcal{T}\mathbf{E},$$

where $\vec{x}$ is a vector of plaintext, $\vec{c}$ is a vector of ciphertext, and $\mathcal{T}$ is a $m \times L$ matrix determined by the seed s, such that[3] $\mathcal{T}_{i,j} = 1$ if $j \in \{t_{i,\iota} : t_{i,\iota}$ is generated from seed $s\}$; otherwise $\mathcal{T}_{i,j} = 0$. Under known-plaintext attack, adversary knows $\vec{c}$, $\vec{x}$, and s (s implies $\mathcal{T}$). To find the master encryption table $\mathbf{E}$, the adversary just solves the linear system of equations $\mathcal{T}\mathbf{E} = \vec{c} - \vec{x}$ with $\mathbf{E}$ as unknowns.

XOR operation can be viewed as addition in $GF(2)$. So the above attack can also be applied to Anderson *et al.* [4].

Solving a large linear equation system.

Linear system over real numbers can be solved efficiently using many methods, including Gaussian elimination and iterative method (e.g. Gauss-Seidel method and the successive over-relaxation method). To solve a linear system with L equations and L unknowns, the complexity of Gaussian elimination is $O(L^3)$, and the successive over-relaxation method is in $O(tL^2)$ where t denotes the number of iterations.

Available tools for solving linear system include commercial softwares like MATLAB [1, 10], and free open source softwares like LAPACK [3], ScaLAPACK and PLAPACK. Particularly, ScaLAPACK and PLAPACK are extensions of LAPACK for distributed-memory systems and are suitable for grid computing.

Note that the encryption/decryption complexity is linear in S. So to achieve fast decryption, it is desireable to have a small S. For small S, the matrix $\mathcal{T}$ is sparse: Every row has exactly S non-zero entries, which are located at column $t_{i,j}$'s, where $1 \le j \le S$ and $t_{i,j}$'s are generated from seed s. For such matrix, the successive over-relaxation method is even more efficient: $O(tLS)$. Furthermore, if the adversary knows a decryption key $\mathbf{D}_u$, which is a good approximation of the unknowns $\mathbf{E}$, then the adversary has a good initial guess for the iterative method. Consequently, the number t of iterations is likely to be very small.

[3]Here, for simplicity, we assume that $t_{i,\iota}$'s generated from the seed s are all distinct.

4.2.2 KPA with Partial Knowledge of Plaintext.

Let the plaintext vector $\vec{x}$, ciphertext vector $\vec{c}$, matrix $\mathcal{T}$, parameter S, encryption key $\mathbf{E}$ and decryption key $\mathbf{D}_u$ of user u be as described in the above subsections, and let us denote with $\vec{x}_u$ the data obtained by decrypting $\vec{c}$ with key $\mathbf{D}_u$. We use the L^2-Norm as the distance function, and denote it with Dist: for two vectors $\vec{x}$ and $\vec{y}$ of the same dimension n, $\mathsf{Dist}(\vec{x}, \vec{y}) \stackrel{def}{=} \sqrt{\sum_{i=1}^{n} (\vec{x}[i] - \vec{y}[i])^2}$.

We now consider an attacker, who has a decryption key $\mathbf{D}_u$ and an approximation $\vec{y}$ of plaintext $\vec{x}$. The decryption table $\mathbf{D}_u$ is generated by independently adding a Gaussian noise with distribution $\mathcal{N}(0, \sigma_1)$ to each entry of encryption table $\mathbf{E}$ as described in [6]. As a result, the watermark, i.e. each component of vector $\vec{x}_u - \vec{x}$, follows a Gaussian distribution $\mathcal{N}(0, \sigma_1 S)$. We assume that each component of vector $\vec{y} - \vec{x}$ follows a Gaussian distribution $\mathcal{N}(0, \sigma_2)$ for some $\sigma_2 < \sigma_1 S$, to reflect the notion that the KPA-adversary has additional knowledge on $\vec{x}$ beyond $\vec{x}_u$. The goal of the attacker is to find a "better" decryption table $\widetilde{\mathbf{D}}$, such that $\mathsf{Dist}(\widetilde{\mathbf{D}}, \mathbf{E}) < \mathsf{Dist}(\mathbf{D}_u, \mathbf{E})$.

The attacker can find such $\widetilde{\mathbf{D}}$ by solving the (over-determined) linear equation system

$$\vec{y} + \mathcal{T}\widetilde{\mathbf{D}} \approx \vec{c},$$

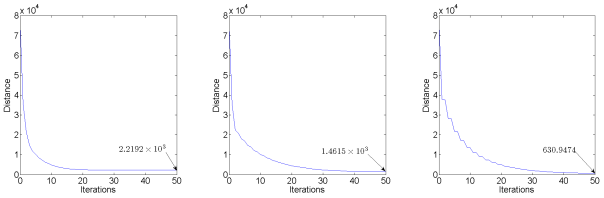
in a least square manner, that is, to find a vector $\widetilde{\mathbf{D}}$ such that the Euclidean length of the vector $(\vec{y} + \mathcal{T}\widetilde{\mathbf{D}} - \vec{c})$ is minimized. This could be solved using least square fitting [12]. Since the attacker already has a decryption table $\mathbf{D}_u$ that is close to the encryption table $\mathbf{E}$, he can use iterative method, for example, the LSQR method[16, 5] and speed up the computation with $\mathbf{D}_u$. When S is small, the matrix $\mathcal{T}$ is sparse, and thus the complexity in solving the linear system can be further reduced [9].

To illustrate the efficiency and effectiveness of the attack, we consider the following setting:

1. The encryption table $\mathbf{E}$ has $2^{16} = 65536$ entries of integers in $\mathbb{Z}_{65536}$.
2. The noise in decryption key $\mathbf{D}_u$ follows distribution $\mathcal{N}(0, \sigma_1)$ with $\sigma_1 = 300$.
3. The noise in vector $\vec{y}$ follows distribution $\mathcal{N}(0, \sigma_2)$ with $\sigma_2 = 30$.
4. The number of equations that the attacker obtained, i.e. $|\vec{y}|$, is $2^{18} = 262144$.
5. We simulate for three different values of S: $S = 4$ (Figure 1(a)), $S = 10$ (Figure 1(b)) and $S = 10$ (Figure 1(c)).

We run the iterative method using MATLAB 7.8.0 in a Windows XP SP3 system on a Dell desktop PC with Intel Core 2 Duo 2.33GHz CPU with 4GB of memory. The simulation utilizes only one CPU core and takes less than 2 hours to finish 50 iterations. Figure 1 shows the distance

$\mathsf{Dist}(\widetilde{\mathbf{D}}, \mathbf{E})$ for different iterations and S, where iteration 0 corresponds to the origin decryption key $\mathbf{D}_u$. From Figure 1, we can see that within 50 iterations, the adversary can obtain a good approximation $\widetilde{\mathbf{D}}$ of the master encryption key $\mathbf{E}$, such that $\mathsf{Dist}(\widetilde{\mathbf{D}}, \mathbf{E}) \leq 0.05 \cdot \mathsf{Dist}(\mathbf{D}_u, \mathbf{E})$.



(a) $S = 4$ (b) $S = 10$ (c) $S = 80$

Figure 1: The distance $\mathsf{Dist}(\widetilde{\mathbf{D}}, \mathbf{E})$ of obtained decryption key $\widetilde{\mathbf{D}}$ from the encryption key $\mathbf{E}$ for 50 iterations for different values of S. Note: (1) The y-axis coordinate corresponding to Iteration 0 indicates $\mathsf{Dist}(\mathbf{D}_u, \mathbf{E})$. (2) The dimensions of vectors $\widetilde{\mathbf{D}}, \mathbf{E}$ and $\mathbf{D}_u$ are all $2^{16} = 65536$.

5. A NEW CHAMELEON ENCRYPTION SCHEME BASED ON MAJORITY VOTE

In this section, we first give an initial construction of our scheme based on majority voting. Then we augment the initial construction with ECC. We defer the analysis of the scheme and the discussion of choice of parameters to Section 6.

5.1 Initial Scheme

Our Chameleon encryption scheme is a one-time pad cipher combined with a special PRNG, called MUTABLE-PRNG.

5.1.1 Mutable Pseudo Random Number Generator.

Let $\mathsf{G} : \{0,1\}^\kappa \to \{0,1\}^{poly(\kappa)}$ be a PRNG , and $\widetilde{\mathsf{G}}(s,i)$ be the i-th bit of the output of $\mathsf{G}(s)$. We construct a PRNG F based on G as follows: Let $s_i \in \{0,1\}^\kappa, 1 \leq i \leq n$, $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ be a (multi)set, and $\|$ denote the string concatenation.

$$\widetilde{\mathsf{F}}(\mathcal{S}, i) \stackrel{def}{=} \texttt{The majority bit among } \widetilde{\mathsf{G}}(s_1, i), \ldots, \widetilde{\mathsf{G}}(s_n, i);$$

$$\mathsf{F}(\mathcal{S}) \stackrel{def}{=} \widetilde{\mathsf{F}}(\mathcal{S}, 1) \| \widetilde{\mathsf{F}}(\mathcal{S}, 2) \| \widetilde{\mathsf{F}}(\mathcal{S}, 3) \| \ldots \| \widetilde{\mathsf{F}}(\mathcal{S}, poly(\kappa)). \quad (1)$$

We call F a MUTABLE-PRNG, since F is a PRNG and satisfies a special property:

- There exists an efficient algorithm, which takes as input $\mathcal{S}$ and $\mathcal{U}$ and outputs $\{\mathcal{S}_u : u \in \mathcal{U}\}$, such that for any u, (1) $\mathcal{S}_u$'s are distinct and (2) $|\mathsf{F}(\mathcal{S}_u)| = |\mathsf{F}(\mathcal{S})|$ and the Hamming distance between $\mathsf{F}(\mathcal{S})$ and $\mathsf{F}(\mathcal{S}_u)$ is small (say, less than 10% of the length of $\mathsf{F}(\mathcal{S})$) with high probability;
- $\mathcal{S}_u$ is short (e.g. $|\mathcal{S}_u| = O(|\mathcal{S}|)$) and the expansion factor of F is large (e.g. $poly(\kappa) = \Omega(\kappa^2)$).

From the security point of view, we require that: Given $\mathcal{S}_u$ and without $\mathcal{S}$, it is hard to find $\mathsf{F}(\mathcal{S}) \oplus \mathsf{F}(\mathcal{S}_u)$ (We defer the security analysis to Section 6.2).

The above property makes F very suitable for watermarking scheme: The content distributor can use seed $\mathcal{S}$ to generate an *encryption key stream*, which will be XORed with the plaintext to generate a ciphertext, and a user u may use seed $\mathcal{S}_u$ to generate a *decryption key stream*, which will be XORed with the ciphertext to generate a watermarked plaintext.

In practice, one may run a secure block cipher (e.g. AES [7]) or hash function (SHA1 [8]) in counter mode to simulate the PRNG G [22] .

5.1.2 The Construction.

1. (Content Distributor) $\mathsf{KGen}(1^\kappa)$: Key setup. Choose n seeds $s_1, s_2, \ldots, s_n$ at random from $\{0,1\}^\kappa$. Keep $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ private as the encryption key. For $u \in \mathcal{U}$, choose at random a subset $\mathcal{S}_u$ of size k from $\mathcal{S}$, and distribute $\mathcal{S}_u$ to user u as the decryption key.
2. (Content Distributor) $\mathsf{Enc}(x; \mathcal{S})$: Encryption of $x = x_1 x_2 x_3 \ldots x_\ell$, where $x_i \in \{0,1\}, 1 \leq i \leq \ell$. The content distributor maintains an integer state variable λ. Initially, $\lambda = 0$. For $1 \leq i \leq \ell$,
$$c_i = x_i \oplus \widetilde{\mathsf{F}}(\mathcal{S}, \lambda + i).$$
Let $c = c_1 c_2 \ldots c_\ell$. Output (c, λ). Update the state variable λ: $\lambda \leftarrow \lambda + \ell$.
3. (User) $\mathsf{Dec}(c, \lambda; \mathcal{S}_u)$: Decryption of (c, λ) using decryption key $\mathcal{S}_u$. For $1 \leq i \leq \ell$,
$$x_{u,i} = c_i \oplus \widetilde{\mathsf{F}}(\mathcal{S}_u, \lambda + i).$$
Output $x_u = x_{u,1} x_{u,2} x_{u,3} \ldots x_{u,\ell}$.

5.2 Extended Scheme with Error Correcting Code

To achieve good quality of the multimedia service from the watermarked data, we have to be able to control the Hamming distance between the watermarked data and the original, or equivalently the Hamming distance between the encryption key stream generated by $\mathsf{F}(\mathcal{S})$ and decryption key stream generated by $\mathsf{F}(\mathcal{S}_u)$. We can adjust this distance by changing the values of k and n. Furthermore, we are able to decrease this distance using ECC, without changing k and n.

The main idea is to divide the data into blocks and apply ECC encoding on them, and then take the resulting codewords as the input of Enc. Formally, let $\mathcal{E}_{\text{ECC}}$ and $\mathcal{D}_{\text{ECC}}$ be the (ℓ, t, d)-error correcting encoding and decoding algorithm based on Hamming distance (e.g. Reed-Solomon Code [18]), where t is the length of the origin code, ℓ is the length of the resulting code, and d is the number of errors that can be corrected. The new encryption and decryption functions are as follows:

$$\mathsf{EncECC}(x, \mathcal{S}) = \mathsf{Enc}(\mathcal{E}_{\text{ECC}}(x), \mathcal{S}); \quad (2)$$

$$\mathsf{DecECC}(c, \mathcal{S}_u) = \mathcal{D}_{\text{ECC}}(\mathsf{Dec}(c, \mathcal{S}_u)). \quad (3)$$

Thus, assuming the bit error rate in the original watermarking scheme $(\mathsf{Enc}, \mathsf{Dec})$ is γ, the block error rate (i.e. the probability that the number of bit errors in a block is more than threshold) of the extended scheme $(\mathsf{EncECC}, \mathsf{DecECC})$ is $1 - \Phi(d; \ell, \gamma)$, where Φ is the Cumulative Distribution Function (CDF) of binomial distribution. Note that when a block is decrypted wrongly, it is likely that some bits are still correct. Thus, the bit error rate of the decrypted message is less than the block error rate.

6. ANALYSIS OF THE NEW CHAMELEON SCHEME

We analyze the Hamming distance between watermarked data and original data in Section 6.1, and evaluate the security in Section 6.2.

6.1 Hamming Distance between the watermarked data and the original

We study the statistical property of the Hamming distance between the decrypted data and the original. For simplicity, we assume that G used in our scheme is an oracle which outputs true random numbers. We first analyze the probability that 1 bit message can be recovered after decryption. Then we suggest typical values for the system parameters in our scheme. After that, we quantify the probability that a data block can be recovered under the typical parameter setting, taking ECC into account. The expected hamming distance between the decrypted data and the original can be derived directly from such probabilities and the length of data.

6.1.1 *Probability to recover 1 bit.*

Let $X_1, X_2, \ldots, X_n$ be n independent random binary variables following Bernoulli distribution with success probability $p = 0.5$. Let $Y_k = \sum_{i=1}^{k} X_i$. Y_k follows Binomial distribution $\mathbf{B}(k, p)$. Let $Z_k = \chi_k(Y_k)$, where $\chi_k(\cdot)$ is defined in Eq (4):

$$\chi_k(x) = \begin{cases} 1 & (\text{If } x \geq \frac{k}{2}); \\ 0 & (\text{otherwise}). \end{cases} \tag{4}$$

Assume the plaintext is just one bit. Z_n represents encryption key stream of one-time pad cipher and Z_k represents the decryption key stream. We intend to quantify the probability $p_{n,k} \stackrel{def}{=} \mathsf{Pr}[Z_k = Z_n] = \mathsf{Pr}[\widetilde{\mathsf{F}}(\mathcal{S}, i) = \widetilde{\mathsf{F}}(\mathcal{S}_u, i)]$ that the decrypted message matches the plaintext, where $u \in \mathcal{U}$ and $1 \leq i \leq poly(\kappa)$. Assume n and k are odd numbers.

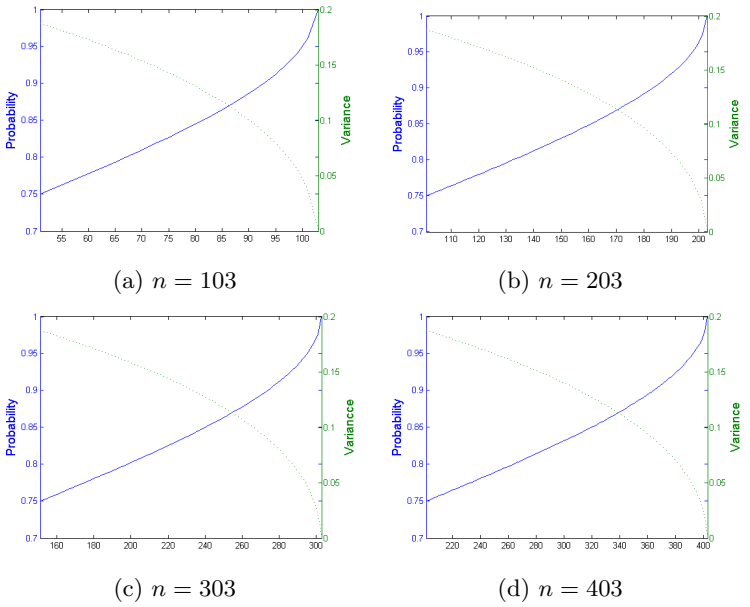
$$\begin{aligned} & \mathsf{Pr}[Z_k = 1, Z_n = 1] \\ = & \sum_{b=(n+1)/2}^{n} \left(\sum_{a=(k+1)/2}^{b} \mathsf{Pr}[Y_k = a,\ Y_n = b] \right) \\ = & \sum_{b=(n+1)/2}^{n} \left(\sum_{a=(k+1)/2}^{b} \mathsf{Pr}[Y_k = a]\mathsf{Pr}[Y_{n-k} = b-a] \right) \\ = & \sum_{b=(n+1)/2}^{n} \left(p^b (1-p)^{n-b} \sum_{a=(k+1)/2}^{b} \binom{k}{a} \binom{n-k}{b-a} \right). \end{aligned}$$

Let $f(n, k, p) \stackrel{def}{=} \mathsf{Pr}[Z_k = 1, Z_n = 1]$. Then $\mathsf{Pr}[Z_k = 0, Z_n = 0] = f(n, k, 1-p)$. Thus, we can express $p_{n,k}$ as a function of n, k and p:

$$\begin{aligned} p_{n,k} &= \mathsf{Pr}[Z_k = 1, Z_n = 1] + \mathsf{Pr}[Z_k = 0, Z_n = 0] \\ &= f(n, k, p) + f(n, k, 1-p). \end{aligned}$$

Let us fix the values of n and p, and view $p_{n,k}$ as a function of k. Figure 2 plots the values of $p_{n,k}$ and $p_{n,k}(1-p_{n,k})$ against k, for different values of n. Note that $p_{n,k}$ is approximately linear in $\frac{k}{n}$.

6.1.2 *Effects of parameters n and k.*



(a) $n = 103$ (b) $n = 203$

(c) $n = 303$ (d) $n = 403$

Figure 2: Plot of expectation $p_{n,k}$ (in solid line) and variance $p_{n,k}(1-p_{n,k})$ (in dashed line) against k, where $p = 0.5$ and $n = 103, 203, 303$ or 403, respectively. For each value of n, the domain of k is all odd numbers within the interval $[\frac{n-1}{2}, n]$.

The parameters n and k can impact the initial construction of scheme in Section 5, in various ways:

1. If n (or k) is even, it is hard to find a proper definition of "majority", such that the majority bit is always unambiguous and unbiased (assuming G outputs true random numbers). Therefore, we recommend that both n and k are odd numbers.
2. Smaller n will result in faster encryption and smaller k will result in faster decryption.
3. The key space size is $\binom{n}{k}$. For a fixed odd number n, $\binom{n}{k}$ achieves maximum when $k = (n-1)/2$. Note that $\binom{2k+1}{k} > 2^k$.
4. The values of n and k determine $p_{n,k}$ which indicates the probability that the decrypted 1-bit message matches the 1-bit plaintext. Figure 2 plots how $p_{n,k}$ varies with value of k for some fixed n.

If we choose $n = 203$ and $k = 101$, then the key space is larger than 2^{101} and the probability $p_{n,k} \approx 0.75$.

6.1.3 *Effect of ECC and Probability to recover a data block.*

We employ ECC to improve similarity between the decrypted message and the plaintext without changing n and k, and present the extended scheme in Section 5.2.

Let us consider the setting $n = 2k + 1$, $n = 203$ and $k = 101$. Then the probability $p_{n,k} \approx 0.75$. We choose ECC code which generate a ℓ bits codeword from t bits plaintext, and can correct up to $(1-\alpha)\ell$ error bits in an ℓ bits codeword. The probability that a block of t bits plaintext could be recovered with probability $1 - \Phi(\alpha\ell; \ell, 0.75)$, where Φ is the Cumulative Distribution Function (CDF) of binomial

distribution $\mathbf{B}(\ell, 0.75)$. We calculate this probability with different values of α and ℓ and show the result in Table 1.

Table 1: The probability that t bits plaintext can be recovered using ECC. In the extended scheme given in Section 5.2, this probability indicates the similarity between the watermarked decrypted message and the original message.

α	0.7	0.7	0.7	0.65	0.65
ℓ	64	128	256	64	128
$1-\Phi(\alpha\ell;\ell,0.75)$	0.8439	0.9058	0.9625	0.9662	0.9933

6.2 Security of the New Chameleon Encryption

6.2.1 Known-Plaintext Attack.

From a plaintext and its ciphertext, one can obtain the key stream used to generate this ciphertext by XORing the plaintext and ciphertext. However, this key stream will not be used any more according to our design. The adversary cannot predict the subsequent bits or guess previous bits from the obtained bit string. That means the obtained key stream cannot help the adversary to decrypt other ciphertexts or to obtain the encryption key. As a result, our scheme is resistant to known-plaintext attack as long as the pseudo random number generator used in our scheme is cryptographically secure. The following Theorem 1 states that the output of $\mathsf{F}(\mathcal{S})$ is unpredictable against adversaries with knowledge of $\mathcal{S}_u \subset \mathcal{S}$. From Theorem 1, the security of our proposed Chameleon encryption scheme against KPA is implied directly. See Corollary 1.

THEOREM 1. *Let F be as constructed in Section 5.1 based on PRNG $\mathsf{G} : \{0,1\}^\kappa \to \{0,1\}^{poly(\kappa)}$, where κ is the security parameter. Let $k > 0$ be an odd integer. Let $\mathcal{S}$ be a (multi)set of $n = 2k+1$ independent random seeds from $\{0,1\}^\kappa$ and $\mathcal{S}_u$ be an arbitrary subset of $\mathcal{S}$ of size k. Suppose G is cryptographically secure. Then, for any PPT algorithm $\mathcal{A}$, which takes as input security parameter κ, the size n of set $\mathcal{S}$, the size k of the subset $\mathcal{S}_u$, the subset $\mathcal{S}_u$, and an i-bit long prefix, denoted as $\mathsf{Prefix}(X,i)$, of $X \leftarrow \mathsf{F}(\mathcal{S})$, and outputs a guess for the next bit $X[i+1]$, for any $1 \le i < poly(\kappa)$, it holds that*

$$\mathsf{Pr}\left[\begin{array}{c}\mathcal{A}(\kappa,n,k,\mathcal{S}_u,\mathsf{Prefix}(X,i)) = X[i+1] \\ \text{where } X \leftarrow \mathsf{F}(\mathcal{S})\end{array}\right] \le \frac{1}{2} + negl(\kappa),$$

where $negl(\cdot)$ is some negligible function. The probability is taken over all random coins of $\mathcal{A}$ and the random coins used to sample $\mathcal{S}$ and $\mathcal{S}_u$.

Note that we adopt the "unpredictability" version of definition of pseudorandomness [11], and unlike the standard setting of pseudorandomness, here adversary $\mathcal{A}$ has access to $\mathcal{S}_u$, which contains partial information of $\mathcal{S}$. The proof of Theorem 1 is given in Appendix A.

COROLLARY 1. *If G is a cryptographically secure PRNG, then the scheme described in Section 5.1 is secure against known-plaintext attack.*

6.2.2 Bound of Information Loss.

Recall that our ECC scheme can correct up to $(1-\alpha)\ell$ error bits at the cost of expending every t-bit message block to ℓ bits codeword. Such error correcting information might be exploited to defer more information about the original plaintext X than he can with the watermarked message $\widetilde{X}$. In addition, recall that our decryption reveals the computation steps, rather than only the outcome, of the majority voting. Thus, the adversary may defer additional information by analyzing the ECC expended ciphertext together with his key $\mathcal{S}_u$. In this section, we quantify this information leakage by analyzing the *average conditional entropy* [19] $\mathbf{H}\big(\widetilde{\mathsf{F}}(\mathcal{S},i) \mid \widetilde{\mathsf{F}}(\mathcal{S}_u,i)\big)$ of the encryption bits, and with that, we bound the remaining entropy $\mathbf{H}\big(X \mid (\mathsf{EncECC}(X,\mathcal{S}),\mathcal{S}_u)\big)$ from below and bound $\mathbf{H}\big(X \mid \widetilde{X}\big) - \mathbf{H}\big(X \mid \mathsf{EncECC}(X,\mathcal{S}),\mathcal{S}_u\big)$ from above.

For simplicity, let us assume G generate true independent unbiased random bits sequence. From the construction of the function F (equation (1)), we will have (detailed computation is given in equation (9) in Appendix B) the entropy of $\mathbf{H}\big(\widetilde{\mathsf{F}}(\mathcal{S},i) \mid \widetilde{\mathsf{F}}(\mathcal{S}_u,i)\big)$ is $-f(n,k,0.5)\log(f(n,k,0.5)) - (1-f(n,k,0.5))\log(1-f(n,k,0.5))$, where $f(n,k,0.5)$ is as define in Section 6.1. For the case $n = 203$ and $k = 101$, we have $f(n,k,0.5) = 0.5p_{n,k} \approx 0.375$ and $\mathbf{H}(\widetilde{\mathsf{F}}(\mathcal{S},i) \mid \widetilde{\mathsf{F}}(\mathcal{S}_u,i)) \approx 0.9544$. Thus, $\mathbf{H}(X|\widetilde{X}) \approx 0.9544t$.

Now we give an upper bound for the overall entropy loss $\mathbf{H}(X) - \mathbf{H}(X \mid \mathsf{EncECC}(X,\mathcal{S}),\mathcal{S}_u)$. Recall the encoding function EncECC in equation (2), we want to quantify the remaining entropy of the original message X given $\mathsf{EncECC}(X,\mathcal{S})$ and $\mathcal{S}_u$. Since X and $\mathcal{S}_u$ are independent, we can view $\widetilde{\mathsf{F}}(\mathcal{S},i) \mid \widetilde{\mathsf{F}}(\mathcal{S}_u,i)$ as the randomness involved in $\mathsf{EncECC}(X,\mathcal{S})$. And note that $\widetilde{\mathsf{F}}(\mathcal{S},i)$ can be recovered from X and $\mathsf{EncECC}(X,\mathcal{S})$. Thus, we can have the the bound for the entropy loss due to observing the ECC code and a decryption key: $\mathbf{H}\big(X\big) - \mathbf{H}\big(X|(\mathsf{EncECC}(X,\mathcal{S}),\mathcal{S}_u)\big)$ is $(1-f(n,k,0.5))\log(1-f(n,k,0.5))$ (detailed computation is given in equation (10) in Appendix B).

For the case $n = 203$ and $k = 101$, $\mathbf{H}(X|\mathsf{EncECC}(X,\mathcal{S}),\mathcal{S}_u)$ is at least $t - 0.0456\ell$ and $\mathbf{H}(X|\widetilde{X}) - \mathbf{H}(X|\mathsf{EncECC}(X,\mathcal{S}),\mathcal{S}_u)$ is at most $0.0456(\ell - t)$.

7. EXTENSION

Our proposed scheme in Section 5 embeds hamming distance based watermark. We may modify it to support L^2-Norm. The modifications are briefed as follows:

1. Redefine $\widetilde{\mathsf{G}}(s,i)$ as the i-th byte of the output of $\mathsf{G}(s,i)$.
2. Redefine $\widetilde{\mathsf{F}}(\mathcal{S},i)$ as

$$\widetilde{\mathsf{F}}(\mathcal{S},i) \stackrel{def}{=} \frac{1}{n}\sum_{j=1}^{n}\widetilde{\mathsf{G}}(s_j,i),$$

where $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$. Namely, replace "majority vote" with "average".

Since s_j's are independent and $\widetilde{\mathsf{G}}(s_j,i)$'s independently identically distributed, we have $\mathsf{E}(\widetilde{\mathsf{F}}(\mathcal{S},i)) = \mathsf{E}(\widetilde{\mathsf{G}}(s_1,i))$ and $\mathsf{Var}(\widetilde{\mathsf{F}}(\mathcal{S},i)) = \frac{1}{n}\mathsf{Var}(\widetilde{\mathsf{G}}(s_1,i))$. Hence, for any subset $\mathcal{S}_u \subset \mathcal{S}$ with size $|\mathcal{S}_u| = k$, $\mathsf{E}(\widetilde{\mathsf{F}}(\mathcal{S}_u,i)) = \mathsf{E}(\widetilde{\mathsf{F}}(\mathcal{S},i))$ and $\mathsf{Var}(\widetilde{\mathsf{F}}(\mathcal{S}_u,i)) = \frac{n}{k}\,\mathsf{Var}(\widetilde{\mathsf{F}}(\mathcal{S},i))$.

8. CONCLUSIONS

We observed that most existing LUT-based Chameleon encryption schemes are vulnerable under known-plaintext attack: Adversary who has partial knowledge of the plaintext, could obtain a good approximation of the encryption key. We proposed a new Chameleon encryption scheme based on a special PRNG which we call MUTABLE-PRNG. We analyzed the scheme and showed that it is secure against known-plaintext attack. Our scheme leaks some information about the plaintexts. We quantified the information leakage by providing an upper bound. How to use cryptographic tools (like homomorphic cryptographic primitives) to avoid information leakage and how to relate conditional entropy to the robustness of watermark directly are in future work.

9. ACKNOWLEDGEMENT

This work is supported by Grant R-252-000-413-232/422/592 from Temasek Defence Systems Institute (TDSI).

10. REFERENCES

[1] MATLAB. www.mathworks.com.

[2] A. Adelsbach, U. Huber, and A. Sadeghi. Fingercasting-Joint Fingerprinting and Decryption of Broadcast Messages. *Transactions on data hiding and multimedia security II*, pages 1–34, 2007.

[3] E. Anderson, C. B. Z. Bai, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen.

[4] R. J. Anderson and C. Manifavas. Chameleon - A New Kind of Stream Cipher. In *International Workshop on Fast Software Encryption*, pages 107–113, 1997.

[5] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. 1994.

[6] M. Celik, A. Lemma, S. Katzenbeisser, and M. van der Veen. Lookup-Table-Based Secure Client-Side Embedding for Spread-Spectrum Watermarks. *IEEE Transactions on Information Forensics and Security*, pages 475–487, 2008.

[7] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. 2002.

[8] D. Eastlake and P. Jones. US secure hash algorithm 1 (SHA1), 2001.

[9] J. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix Anal. Appl*, 13(1):333–356, 1992.

[10] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in matlab: design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356, 1992.

[11] O. Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, New York, NY, USA, 2006.

[12] R. Hanson. *Solving least squares problems*. 1995.

[13] D. Kundur and K. Karthik. Video fingerprinting and encryption principles for digital rights management. In *Proceedings of the IEEE*, pages 918–932, 2004.

[14] A. Lemma, S. Katzenbeisser, M. Celik, and M. Van Der Veen. Secure Watermark Embedding through Partial Encryption. In *International Workshop on Digital Watermarking*, pages 433–445, 2006.

[15] W. Luh and D. Kundur. New paradigms for effective multicasting and fingerprinting of entertainment media. *IEEE Communications Magazine* , pages 77–84, 2005.

[16] C. Paige and M. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1):43–71, 1982.

[17] R. Parviainen and R. Parnes. Large Scale distributed watermarking of multicast media through encryption. In *International Conference on Communications and Multimedia Security Issues*, page 17, 2001.

[18] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, pages 300–304, 1960.

[19] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, pages 379–423, 1948.

[20] P. Tomsich and S. Katzenbeisser. Copyright Protection Protocols For Multimedia Distribution Based On Trusted Hardware. In *Protocols for Multimedia Systems*, pages 815–819, 2000.

[21] P. Tomsich and S. Katzenbeisser. Towards a Secure and De-centralized Digital Watermarking Infrastructure for the Protection of Intellectual Property. In *EC-Web*, pages 38–47, 2000.

[22] A. Young and M. Yung. *Malicious Cryptography: Exposing Cryptovirology*. 2004.

APPENDIX

A. PROOF OF THEOREM 1

Recall that in Section 5.1, we defined functions $\widetilde{\mathsf{G}}(\cdot,\cdot)$, $\mathsf{F}(\cdot)$, and $\widetilde{\mathsf{F}}(\cdot,\cdot)$, from some PRNG $\mathsf{G}(\cdot)$. For the sake of proof, let us define two functions $\widetilde{\mathcal{F}}$ and $\mathcal{F}$, in a similar way that we defined $\widetilde{\mathsf{F}}$ and F. Let set $\mathcal{S} = \{s_1, \ldots, s_n\}$. If n is even and $\sum_{i=1}^{n} \widetilde{\mathsf{G}}(s_i, j) = n/2$, $\widetilde{\mathcal{F}}(\mathcal{S}, j)$ outputs a random bit, otherwise $\widetilde{\mathcal{F}}(\mathcal{S}, j) = \widetilde{\mathsf{F}}(\mathcal{S}, j)$. $\mathcal{F}(\mathcal{S}) = \widetilde{\mathcal{F}}(\mathcal{S}, 1) \| \widetilde{\mathcal{F}}(\mathcal{S}, 2) \| \ldots$ Note that $\widetilde{\mathcal{F}}(\mathcal{S}, j)$ outputs an unbiased but potentially ambiguous majority bit.

In order to prove Theorem 1, we introduce and prove the following claim.

CLAIM 1. *Let F be as constructed in Section 5.1 based on PRNG $\mathsf{G} : \{0,1\}^{\kappa} \to \{0,1\}^{poly(\kappa)}$, where κ is the security parameter. Let k be an odd integer. Let $\mathcal{T}$ be a (multi)set of $(k+1)$ independent random seeds from $\{0,1\}^{\kappa}$. If G is cryptographically secure, then for any PPT algorithm $\mathcal{B}$, for any $1 \leq i < poly(\kappa)$, it holds that*

$$\mathsf{Pr}\left[\begin{array}{c} \mathcal{B}(\kappa, k, 1^{|Y|}, y_1, \ldots, y_i) = Y[i+1] \\ \text{where } Y \leftarrow \mathcal{F}(\mathcal{T}) \text{ and } \forall 1 \leq j \leq i, \\ y_j \leftarrow \sum_{s \in \mathcal{T}} \widetilde{\mathsf{G}}(s, j) \end{array}\right] \leq \frac{1}{2} + negl(\kappa),$$

where $negl(\cdot)$ denotes some negligible function and the probability is taken over all random coins of $\mathcal{B}$ and the random coins used to sample $\mathcal{T}$.

PROOF SKETCH OF CLAIM 1. Suppose there exists a PPT algorithm $\mathcal{B}$ which can break Claim 1 and guess the next bit correctly with probability $1/2 + \epsilon_1$. Then we can construct a PPT algorithm $\mathcal{C}$ which can break the pseudorandomness of

G, i.e. for some i^*, let U_κ denote a uniform random variable over $\{0,1\}^\kappa$, for non-negligible function $\epsilon_2(\kappa)$,

$$\mathsf{Pr}\left[\begin{array}{c} \mathcal{C}(\kappa, 1^{|X|}, \mathsf{Prefix}(X, i^*)) = X[i^*+1] \\ \text{where } X \leftarrow \mathsf{G}(U_\kappa) \end{array}\right] \geq \frac{1}{2} + \epsilon_2(\kappa). \tag{5}$$

Construction of algorithm $\mathcal{C}$.

1. The input of $\mathcal{C}$ is the security parameter κ, the length of X, and i^*-bit prefix of X.
2. Choose k elements $s_1, \ldots, s_k$ from $\{0,1\}^\kappa$ independently and uniformly randomly. Let (multi)set $\mathcal{T}_1 = \{s_1, \ldots, s_k\}$.
3. For each j, $1 \leq j \leq i^*+1$, compute $u_j \leftarrow \sum_{s \in \mathcal{T}_1} \widetilde{\mathsf{G}}(s, j)$.
4. For each j, $1 \leq j \leq i^*$, let σ_j be the j-th bit of $\mathsf{Prefix}(X, i^*)$ and set $y_j \leftarrow u_j + \sigma_j$.
5. Invoke algorithm $\mathcal{B}$ on input $(\kappa, k, 1^{|X|}, y_1, \ldots, y_{i^*})$ and obtain output $\zeta \in \{0,1\}$.
6. If $(k+1)/2 - 1 \leq u_{i^*+1} \leq (k+1)/2$, output ζ; otherwise output a random bit.

One can verify that the constructed algorithm $\mathcal{C}$ satisfies equation (5) and breaks the pseudorandomness of G with non-negligible advantage. Here we save the details. □

The proof for Theorem 1 is given below.

Proof of Theorem 1. Suppose that there exists a PPT algorithm $\mathcal{A}$ which has probability $\frac{1}{2} + \epsilon$ to guess the next bit correctly. More precisely, for some i^*, $1 \leq i^* < poly(\kappa)$, and some non-negligible function ϵ in κ,

$$\mathsf{Pr}\left[\begin{array}{c} \mathcal{A}(\kappa, n, k, \mathcal{S}_u, \mathsf{Prefix}(X, i^*)) = X[i^*+1] \\ \text{where } X \leftarrow \mathsf{F}(\mathcal{S}) \end{array}\right] = \frac{1}{2} + \epsilon.$$

We intend to construct a PPT algorithm $\mathcal{B}$ based on $\mathcal{A}$, such that $\mathcal{B}$ can break Claim 1 with probability about $\frac{1}{2} + \frac{1}{4}\epsilon$: Let $\mathcal{T} = \mathcal{S} \setminus \mathcal{S}_u$,

$$\mathsf{Pr}\left[\begin{array}{c} \mathcal{B}(\kappa, k, 1^{|Y|}, y_1, \ldots, y_{i^*}) = Y[i^*+1] \\ \text{where } Y \leftarrow \mathcal{F}(\mathcal{T}) \text{ and } \forall 1 \leq j \leq i^*, \\ y_j \leftarrow \sum_{s \in \mathcal{T}} \widetilde{\mathsf{G}}(s, j) \end{array}\right] \approx \frac{1}{2} + \frac{1}{4}\epsilon.$$

Construction of Adversary $\mathcal{B}$ against Claim 1.

1. The input of $\mathcal{B}$ is $(\kappa, 1^{|Y|}, y_1, \ldots, y_{i^*})$ where $\forall 1 \leq j \leq i^*, y_j \leftarrow \sum_{s \in \mathcal{T}} \widetilde{\mathsf{G}}(s, j)$, and $\mathcal{T}$ is a set of $(k+1)$ independent random elements from $\{0,1\}^\kappa$.
2. Choose k elements $s_1, \ldots, s_k$ independently and uniformly randomly from $\{0,1\}^\kappa$. Let $\mathcal{T}' = \{s_1, \ldots, s_k\}$.
3. For each $1 \leq j \leq i^*$, compute $z_j \leftarrow \sum_{s \in \mathcal{T}'} \widetilde{\mathsf{G}}(s, j)$.
4. For each $1 \leq j \leq i^*$, if $y_j + z_j \geq k+1$, then set $\nu_j \leftarrow 1$, otherwise set $\nu_j \leftarrow 0$.
5. Invoke $\mathcal{A}$ on input $(\kappa, 2k+1, k, \mathcal{T}', \nu_1 \nu_2 \ldots \nu_{i^*})$, and obtain output $b \in \{0,1\}$.
6. Compute $b' \leftarrow \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1) \quad \in \{0,1\}$.
7. If $b \neq b'$, then output b; otherwise output a random bit.

Success Probability of Adversary $\mathcal{B}$ against Claim 1.

Let $\mathbf{E}_\mathcal{A}$ ($\mathbf{E}_\mathcal{B}$, respectively) denote the event that $\mathcal{A}$ ($\mathcal{B}$, respectively) guesses the next bit correctly. The input that $\mathcal{B}$ feeds in $\mathcal{A}$ has identical distribution as the input of $\mathcal{A}$ in Theorem 1. Therefore, according to our hypothesis in the beginning of proof, $\mathsf{Pr}[\mathbf{E}_\mathcal{A}] = \frac{1}{2} + \epsilon$.

Let $\mathcal{W} = \mathcal{T} \cup \mathcal{T}'$, and $\mathsf{Pr}\left[\widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\right] = q$. We have to point out that through our calculation in Section 6.1 (See Figure 2), $q \approx 0.75$, and we will substitute q with 0.75 in the last step of calculation in order to avoid the potential amplification of numerical errors during the calculation.

Our calculation will utilize these facts:

1. $\mathsf{Pr}[b = \widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) \mid \mathbf{E}_\mathcal{A}] = 1$.
2. $\mathsf{Pr}[b = b' \mid \mathbf{E}_\mathcal{A}] = \mathsf{Pr}[b \neq b' \mid \neg\mathbf{E}_\mathcal{A}] = \mathsf{Pr}\left[\widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\right] = q$.
3. $\mathsf{Pr}[\mathbf{E}_\mathcal{B} \mid \mathbf{E}_\mathcal{A}, b \neq b'] = 1$.

Now we calculate the probability $\mathsf{Pr}[\mathbf{E}_\mathcal{B}]$ that $\mathcal{B}$ guesses the next bit correctly.

$$\begin{aligned}
& \mathsf{Pr}[\mathbf{E}_\mathcal{B}] \\
= \; & \mathsf{Pr}\left[\mathbf{E}_\mathcal{B} \mid b = b'\right] \mathsf{Pr}\left[b = b'\right] + \mathsf{Pr}\left[\mathbf{E}_\mathcal{B}, b \neq b'\right] \\
= \; & \frac{1}{2}\Big(\mathsf{Pr}\left[b = b' \mid \mathbf{E}_\mathcal{A}\right] \mathsf{Pr}[\mathbf{E}_\mathcal{A}] + \mathsf{Pr}\left[b = b' | \neg\mathbf{E}_\mathcal{A}\right] \mathsf{Pr}[\neg\mathbf{E}_\mathcal{A}]\Big) \\
& + \Big(\mathsf{Pr}\left[\mathbf{E}_\mathcal{B} \mid \mathbf{E}_\mathcal{A}, b \neq b'\right] \mathsf{Pr}\left[b \neq b' \mid \mathbf{E}_\mathcal{A}\right] \mathsf{Pr}[\mathbf{E}_\mathcal{A}] + \\
& \mathsf{Pr}\left[\mathbf{E}_\mathcal{B} \mid \neg\mathbf{E}_\mathcal{A}, b \neq b'\right] \mathsf{Pr}\left[b \neq b' \mid \neg\mathbf{E}_\mathcal{A}\right] \mathsf{Pr}[\neg\mathbf{E}_\mathcal{A}]\Big) \\
= \; & \frac{1}{2}\Big(q \cdot (\frac{1}{2} + \epsilon) + (1-q) \cdot (\frac{1}{2} - \epsilon)\Big) + \\
& \Big(1 \cdot (1-q) \cdot (\frac{1}{2} + \epsilon) + \\
& \mathsf{Pr}\left[\mathbf{E}_\mathcal{B} \mid \neg\mathbf{E}_\mathcal{A}, b \neq b'\right] \cdot q \cdot (\frac{1}{2} - \epsilon)\Big)
\end{aligned} \tag{6}$$

We turn to calculate the probability $\mathsf{Pr}[\mathbf{E}_\mathcal{B} \mid \neg\mathbf{E}_\mathcal{A}, b \neq b']$.

$$\begin{aligned}
& \mathsf{Pr}\left[\mathbf{E}_\mathcal{B} \mid \neg\mathbf{E}_\mathcal{A}, b \neq b'\right] \\
= \; & \frac{\mathsf{Pr}[\mathbf{E}_\mathcal{B}, b \neq b' \mid \neg\mathbf{E}_\mathcal{A}]}{\mathsf{Pr}[b \neq b' \mid \neg\mathbf{E}_\mathcal{A}]} \\
= \; & \frac{\mathsf{Pr}\left[\widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) \neq \widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\right]}{q} \\
= \; & \frac{1}{q}\Big(\mathsf{Pr}\left[\widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\right] - \\
& \mathsf{Pr}\left[\widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\right]\Big) \\
= \; & 1 - \frac{1}{q}\mathsf{Pr}\left[\widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\right]
\end{aligned} \tag{7}$$

Now we turn to calculate the probability $\mathsf{Pr}\Big[\widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\Big]$. At first, we define k' as

$$k' = \begin{cases} k+1 & \text{(If } k \text{ is odd);} \\ k & \text{(otherwise).} \end{cases}$$

Note that $\mathcal{W} = \mathcal{T} \cup \mathcal{T}'$, $\widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1) \Rightarrow \widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathcal{F}}(\mathcal{T}, i^*+1)$.

$$
\begin{aligned}
& \mathsf{Pr}\left[\widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\right] \\
= \; & \mathsf{Pr}\left[\widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\right] \\
= \; & \sum_{j\in\{0,1\}} \mathsf{Pr}\left[\widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) = j, \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1) = j\right] \\
= \; & \sum_{j\in\{0,1\}} \mathsf{Pr}\left[\widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) = j\right] \mathsf{Pr}\left[\widetilde{\mathsf{F}}(\mathcal{T}', i^*+1) = j\right] \\
= \; & \sum_{j\in\{0,1\}} \frac{1}{2}\cdot\frac{1}{2} \\
= \; & \frac{1}{2}. \qquad (8)
\end{aligned}
$$

Substituting $\mathsf{Pr}\left[\widetilde{\mathsf{F}}(\mathcal{W}, i^*+1) = \widetilde{\mathcal{F}}(\mathcal{T}, i^*+1) = \widetilde{\mathsf{F}}(\mathcal{T}', i^*+1)\right] = \frac{1}{2}$ into equation (7), we have $\mathsf{Pr}\left[\mathbf{E}_{\mathcal{B}} \mid \neg\mathbf{E}_{\mathcal{A}}, b \neq b'\right] = 1 - \frac{1}{2q}$. Substituting this result into equation (6), we have

$$\mathsf{Pr}\left[\mathbf{E}_{\mathcal{B}}\right] = \frac{1}{2} + \epsilon(1-q) \approx \frac{1}{2} + \frac{1}{4}\epsilon.$$

Since ϵ is non-negligible in κ, $\mathsf{Pr}\left[\mathbf{E}_{\mathcal{B}}\right] - \frac{1}{2} = \epsilon(1-q) \approx \frac{1}{4}\epsilon$ is non-negligible in κ.

Therefore, $\mathcal{B}$ breaks Claim 1. The contradiction implies that our hypothesis in the beginning of this proof is false and consequently Theorem 1 is correct. □

B. MORE DETAILS ON THE COMPUTATION OF BOUND OF INFORMATION LOSS

We will now give the computation for the entropy of $\mathbf{H}\big(\widetilde{\mathsf{F}}(\mathcal{S}, i) \mid \widetilde{\mathsf{F}}(\mathcal{S}_u, i)\big)$:

$$
\begin{aligned}
& \mathbf{H}\big(\widetilde{\mathsf{F}}(\mathcal{S}, i) \mid \widetilde{\mathsf{F}}(\mathcal{S}_u, i)\big) \\
= & -\sum_{y\in\{0,1\}} \frac{1}{2}\cdot \sum_{x\in\{0,1\}} \Big(\mathsf{Pr}\big(\widetilde{\mathsf{F}}(\mathcal{S}_u, i) = y \mid \widetilde{\mathsf{F}}(\mathcal{S}, i) = x\big)\cdot \\
& \qquad \log \mathsf{Pr}\big(\widetilde{\mathsf{F}}(\mathcal{S}_u, i) = y \mid \widetilde{\mathsf{F}}(\mathcal{S}, i) = x\big)\Big) \\
= & -f(n,k,0.5)\log(f(n,k,0.5)) - (1-f(n,k,0.5))\log(1-f(n,k,0.5)) \qquad (9)
\end{aligned}
$$

From $\mathbf{H}\big(\widetilde{\mathsf{F}}(\mathcal{S}, i) \mid \widetilde{\mathsf{F}}(\mathcal{S}_u, i)\big)$, we can have:

$$
\begin{aligned}
& \mathbf{H}\big(x\big) - \mathbf{H}\big(x|(\mathsf{EncECC}(x,\mathcal{S}), \mathcal{S}_u)\big) \\
\leq \; & \mathbf{H}\big(x\big) - \Big(\mathbf{H}\big(x\big) - \log(2^{\ell}) + \ell\mathbf{H}\big(\widetilde{\mathsf{F}}(\mathcal{S}, i)|\widetilde{\mathsf{F}}(\mathcal{S}_u, i)\big)\Big) \\
= \; & \ell + \ell\cdot\big(\; f(n,k,0.5)\log(f(n,k,0.5)) + \\
& (1-f(n,k,0.5))\log(1-f(n,k,0.5))\;\big). \qquad (10)
\end{aligned}
$$

where the inequality holds since the information carried by $\mathsf{EncECC}(x,\mathcal{S})$ is at most ℓ bits, and we can view $\ell\mathbf{H}\big(\widetilde{\mathsf{F}}(\mathcal{S}, i)|\widetilde{\mathsf{F}}(\mathcal{S}_u, i)\big)$ as the randomness that can be recovered by x, $\mathsf{EncECC}(x,\mathcal{S})$ and $\mathcal{S}_u$).