ANONYMOUS INCOMPRESSIBLE ENCRYPTION

Thesis submitted by

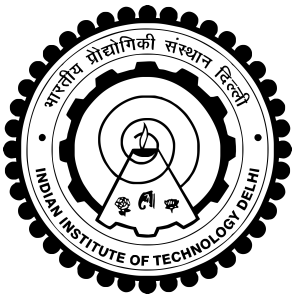
Abhinav Kumar
2019CS50415

under the guidance of

Prof. Venkata Koppula, Indian Institute of Technology Delhi

in partial fulfilment of the requirements
for the award of the degree of

Bachelor and Master of Technology



Department Of Computer Science & Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI

June 2024

THESIS CERTIFICATE

This is to certify that the thesis titled **Anonymous Incompressible Encryption**, submitted by **Abhinav Kumar (2019CS50415)**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Bachelor+Masters of Technology**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Venkata Koppula
Assistant Professor
Department of CSE
IIT Delhi
New Delhi, India 110016

Date: June 2024

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor Prof. Venkata Koppula for his encouragement, patience, invaluable advice, and immense guidance throughout this journey. This thesis would not have been possible without his support.

My sincere thanks to Mahesh Sreekumar Rajasree for always being ready for discussions, always being helpful, and for his unwavering assistance in clearing my doubts.

I am also immensely grateful to my friends, especially Aman Verma, Dhairya Gupta and Pratyush Saini, for standing by my side, listening to me during difficult times, and helping me navigate these past few years.

Last but not least, I extend my heartfelt thanks to my wonderful family for their unwavering support through thick and thin. I would not be where I am today without them.

Abhinav Kumar

ABSTRACT

In the realm of theoretical cryptography, achieving robust security has led to various definitions for secret key (SKE) and public key encryption (PKE). CPA and CCA security are gold standards, but they sometimes fall short as cryptographic applications evolve. One of the shortcomings is that the public key is anonymous. CPA security ensures that the message is hidden but it does not guarantee that the ciphertext conceals the public key(or the receiver's identity). Bellare et al. introduced the concept of anonymous encryption to counter this. Private key security models are widely understood, but usually, these models presuppose that the secret key cannot be disclosed. If the secret key is leaked, an adversary with only a short digest of the ciphertext can still preserve semantic security. This notion, called incompressible encryption, was initially proposed by Dziembowski for SKE and later extended to PKE by Guan, Wichs, and Zhandry. Incompressible encryption strengthens CPA security through a security game where the adversary compresses the ciphertext before obtaining the secret key.

This thesis extends incompressible encryption to include anonymity. While typically associated with public-key settings, anonymity is also relevant in incompressible secret-key settings where the secret key may be obtained later. This work aims to develop more robust encryption schemes for modern cryptographic applications.

KEYWORDS: anonymity, incompressible-encryption

Contents

ABBREVIATIONS

IITD	Indian Institute of Technology, Delhi
PPT	Probabilistic Polynomial Time
SKE	Secret Key Encryption
PKE	Public Key Encryption
CPA	Chosen Plaintext Attack
CCA	Chosen Ciphertext Attack
PRF	Pseudorandom Functions
PPRF	Programmable Pseudorandom Functions
HPS	Hash Proof System

NOTATION

λ	Security parameter
$\mathbf{negl}(\cdot)$	Negligible function in the input
n	Message size
$\bar{0}$	All zeros
S	S-bit digest used in incompressible encryption schemes

Chapter 1

INTRODUCTION

Cryptography is the science of securing communication and data against unauthorized access and manipulation. Its main goal is to ensure that information remains confidential, integral, and authentic even in the presence of adversaries. This is done using cryptographic algorithms and protocols that transform plaintext information into encrypted data, that is only accessible to intended recipients having the correct key. Encryption is one of the basic concepts in cryptography. It is the process of converting plain text into an unreadable version called ciphertext using an algorithm and encryption key. Decryption is just the reverse of encryption, where the ciphertext is converted back into plain text using a decryption key. There are two main types of encryption: secret key encryption (SKE) and public key encryption (PKE).

Secret Key Encryption: SKE, which is also known as symmetric encryption, uses the same key for both encryption and decryption. This implies that both the sender and the receiver have to have the same secret key and should not disclose it to anyone. Even though SKE is computationally efficient and simple, it has problems in securely transferring the key between the two parties, particularly through an insecure channel.

Public Key Encryption: PKE, which is also known as asymmetric encryption, addresses the key distribution problem by using two different keys: a public key and a secret key. The public key is made available to the public and is used in the encryption process while the secret key is kept secure and it is used in the decryption process. This makes it possible for a sender to encrypt the message and send it to a receiver without having to first exchange secret keys. This method simplifies key distribution but is computationally more intensive. PKE is the foundation of most of the contemporary cryptographic procedures and is crucial for safe communication over the internet.

1.1 Security Models

As cryptographic techniques have evolved, so have the methods attackers use to break them. To counteract these threats, the cryptographic community has established several security definitions. Two widely recognized are security against chosen plaintext attacks (CPA) [GM82] and security against chosen ciphertext attacks (CCA) [NY90, RS91].

CPA Security: In a chosen plaintext attack, the adversary can choose arbitrary plaintexts and obtain their corresponding ciphertexts. CPA security ensures that even with this capability, the adversary cannot learn anything about the plaintexts beyond what they could

guess without those ciphertexts. Essentially, CPA security guarantees that the encryption scheme is semantically secure, meaning it is computationally infeasible for an adversary to distinguish between the encryptions of any two chosen plaintexts.

The CPA security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is defined as follows:

- **Initialization phase:**
  - The challenger $\mathcal{C}$ generates a key k using the key generation algorithm KeyGen and also generates a random bit $b \in \{0, 1\}$.
- **Query Phase:**
  - The adversary $\mathcal{A}$ can query the challenger $\mathcal{C}$ with any plaintext m of its choice.
  - The challenger responds with the ciphertext $\mathsf{ct} = \mathsf{Enc}(k, \mathsf{m})$.
- **Challenge Phase:**
  - The adversary $\mathcal{A}$ sends two plaintexts m_0 and m_1 to $\mathcal{C}$.
  - The challenger $\mathcal{C}$ computes the challenge ciphertext as $\mathsf{ct}^* = \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_b)$.
  - The ciphertext ct^* is sent to the adversary $\mathcal{A}$.
- **Guess Phase:**
  - The adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ for the value of b.
  - $\mathcal{A}$ wins the game if $b' = b$.

The encryption scheme is considered CPA secure if there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all λ,

$$\mathrm{Adv}_{\mathcal{A}} = \left| \Pr[\mathcal{A} \text{ wins }] - \frac{1}{2} \right| \leq \mathrm{negl}(\lambda)$$

CCA Security: A chosen ciphertext attack is more powerful than a CPA, as it allows the adversary to choose ciphertexts and obtain their corresponding plaintexts, except for the challenge ciphertext that they are trying to break. CCA security provides a stronger guarantee by ensuring that even with this additional capability, the adversary cannot decrypt the challenge ciphertext or gain any useful information from it. CCA-secure schemes are designed to resist adaptive attacks, making them robust against a broader range of potential threats.

The CCA security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is defined as follows:

- **Initialization phase:**
  - The challenger $\mathcal{C}$ generates a key k using the key generation algorithm KeyGen and also samples a random bit $b \in \{0, 1\}$.
- **Pre-Challenge Query Phase:** The adversary $\mathcal{A}$ sends polynomially many queries. Each query can either be an encryption query or a decryption query.
  - **Encryption Query:** The adversary $\mathcal{A}$ sends a message m. The challenger responds with the ciphertext $\mathsf{ct} = \mathsf{Enc}(k, m)$.

  - **Decryption Query:**The adversary sends any ciphertext ct' to receive the corresponding message $\mathsf{m}' = \mathsf{Dec}(k, \mathsf{ct}')$.

- **Challenge Phase:**
  - The adversary $\mathcal{A}$ sends two messages m_0 and m_1.
  - The challenger $\mathcal{C}$ computes the challenge ciphertext $\mathsf{ct}^* = \mathsf{Enc}(k, \mathsf{m}_b)$.
  - The ciphertext ct^* is sent to the adversary $\mathcal{A}$.

- **Post-Challenge Query Phase:** The adversary $\mathcal{A}$ sends polynomially many queries. Each query can either be an encryption query or a decryption query.
  - **Encryption Query:** The adversary $\mathcal{A}$ sends a message m. The challenger responds with the ciphertext $\mathsf{ct} = \mathsf{Enc}(k, m)$.
  - **Decryption Query:**The adversary sends any ciphertext $\mathsf{ct}'(\neq \mathsf{ct}^*)$ to receive the corresponding message $\mathsf{m}' = \mathsf{Dec}(k, \mathsf{ct}')$.

- **Guess Phase:**
  - The adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ for the value of b.
  - The adversary $\mathcal{A}$ wins the game if $b' = b$.

The encryption scheme is considered CCA secure if there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all λ,

$$\mathrm{Adv}_{\mathcal{A}} = \left| \Pr[\mathcal{A} \text{ wins }] - \frac{1}{2} \right| \leq \mathrm{negl}(\lambda)$$

Similarly, we can define these CPA and CCA games for the PKE setting.

1.2 Anonymous Encryption

While CPA and CCA security are considered the gold standards in cryptographic security, evolving applications and threats have revealed their limitations. One such limitation is the inability to guarantee the anonymity of the public key in PKE schemes. This has led to the development of encryption schemes that ensure that the ciphertext does not reveal the public key used (that is, the intended recipient of the message). Encryption schemes that fulfill this requirement are called *anonymous encryption schemes*.

Anonymity in encryption is crucial in scenarios where the mere knowledge of the recipient can compromise privacy or security. For example, in anonymous communication systems or whistleblower platforms, the ability to conceal the recipient's identity is as important as hiding the message content. Bellare et al. [BBDP01] highlighted the importance of public key anonymity and proposed encryption schemes that address this requirement.

The above security notion is now very well studied, and we have several efficient constructions for anonymous encryption [BBDP01, HSHI02, ABN10, Moh10, HLH+22, GMP22], based on different computational assumptions.

1.3 Incompressible Encryption

Another advanced concept in cryptography is incompressible encryption. This concept extends traditional encryption security to scenarios where the adversary eventually learns the secret key. This concept of incompressible encryption means that even if the adversary gets the secret key and a short digest of the ciphertext, they cannot produce the original message. Studied by Dziembowzki [Dzi06] for SKE, who proposed the notion of *forward secure storage*. Guan, Wichs, and Zhandry [GWZ22] extended this idea to the public key setting, introducing the concept of incompressible encryption.

Incompressibility strengthens CPA security by incorporating an additional challenge: the adversary must compress the ciphertext into a short digest before receiving the secret key and then attempt to guess the original message. This security notion is particularly relevant in scenarios where the secret key may eventually be exposed, but the ciphertexts remain secure as long as they are not stored in full. Similarly, one can define incompressible CCA security.

An important parameter in incompressible encryption schemes is the *rate*, defined as the ratio of the size of the message to the size of the ciphertext. Recent research [GWZ22, BDD22, GWZ23] has focused on developing rate-1 CPA/CCA secure incompressible encryption schemes, where the ciphertext size is close to the message size, ensuring efficiency without compromising security.

In this work, we extend the concept of an incompressible encryption scheme to anonymity. While anonymity is usually studied in the public key setting, it's notable that in the incompressible secret-key setting, anonymity remains relevant since adversaries can later obtain the secret key used in encryption.

1.4 Results

We outline the various schemes proposed in this thesis. We proved that the incompressible SKE schemes proposed by Dziembowski [Dzi06] exhibits anonymity. This gives us two anonymous SKE schemes secure against adversaries that make a single encryption query. We also show that the scheme can be modified using programmable PRFs to achieve anonymous incompressible security against unbounded encryption queries. Notably, this is the first non-trivial[1] incompressible SKE scheme that can handle unbounded encryption queries.

We demonstrate that the rate-1 incompressible SKE scheme proposed by Branco *et al.* [BDD22] can be modified to achieve anonymity and security against CPA. Furthermore,

[1]One can certainly use an incompressible PKE scheme (for example, the ones proposed by Guan et al. [GWZ22] and Branco et al. [BDD22]) to build an SKE scheme that handles unbounded encryption queries.

we show that the hybrid encryption proposed by Branco *et al.* that achieves rate-1 CPA secure incompressible PKE using programmable HPS can achieve anonymity provided the underlying HPS and incompressible SKE schemes are anonymous.

This work is a part of a joint effort as detailed in Koppula *et al.* [KKRS24].

1.5 Related Works in the Incompressible Setting

Dziembowski [Dzi06] pioneered the first incompressible encryption scheme, a SKE scheme that offers security against unbounded adversaries. This foundational work laid the groundwork for subsequent research in the field. More recently, Guan et al. [GWZ22] introduced the first incompressible PKE schemes. Their initial construction was based on any CPA-secure PKE scheme but had a low rate, limiting its practical efficiency. To address this, they developed a second construction based on indistinguishability obfuscation (iO) [GGH+16, JLS21], achieving a rate of 1.

Further advancements were made by Branco et al. [BDD22], which was built on the concept of incompressible encodings [MW20] and variants of hash proof systems. They successfully created rate-1 constructions for both incompressible SKE and CCA secure incompressible PKE schemes. Additionally, they proposed a scheme within the ideal cipher model, broadening the scope of their constructions.

Expanding the application of incompressible encryption, Guan et al. [GWZ23] explored its extension to a multi-user environment. They proposed schemes that operate within the random oracle model, demonstrating the versatility and potential of incompressible encryption in diverse cryptographic contexts.

In recent development, [GKRV24] proposed an extension of incompressibility to the functional encryption. Their work provides incompressible attribute-based encryption for circuits based on standard assumptions, achieving optimal efficiency with short secret keys. Additionally, they propose incompressible functional encryption for circuits obtained from non-incompressible functional encryption, offering two variants: the one where the secret keys are short one with relatively small secret keys, and another with large secret keys. These advancements represent a significant leap in the efficiency and practical applicability of incompressible encryption.

1.6 Open Problems

This work initiates the study of anonymity in the incompressibility setting. Below, we list a few interesting open questions:

- Do we need programmable PRFs to construct incompressible SKE that can handle

unbounded encryption queries? In particular, can we build incompressible SKE that can handle unbounded encryption queries, using only one way functions?

- Can we construct a rate-1 anonymous incompressible secret key encryption (SKE) scheme in the random oracle model?

Chapter 2

Technical Overview

In this technical overview, we present a sketch of the constructions and offer a high-level proof of their security. Our inspiration for these schemes stems from Dziembowski's construction [Dzi06], so we commence with a brief overview of his scheme.

Dziembowski's Incompressible SKE Scheme. The secret key consists of a pair of strings (s_1, s_2) where s_1 is a random seed for an extractor Ext, and s_2 is a random string of the same length as the message to be encrypted. During encryption, a random string R is generated and applied to the extractor to obtain $k' \leftarrow \mathsf{Ext}(R; s_1)$. A *one-time* key k is generated by XORing k' with s_2. This key is then used to encrypt the message m, resulting in the ciphertext (R, t) where $t = m \oplus k$.

To demonstrate security against unbounded adversaries, the proof employs a series of hybrid games. In the first hybrid game, the generation of the secret key is delayed. Instead, both R and t are generated randomly. Later, to derive the secret key (s_1, s_2), s_1 is chosen randomly while s_2 is set to $\mathsf{Ext}(R; s_1) \oplus t \oplus m$. It is evident that this hybrid game is identical to the original incompressible security game.

In the subsequent hybrid, s_2 is chosen uniformly at random. The indistinguishability argument relies on the properties of the extractor Ext. Specifically, since R is random and independent of s_1, the value $k' = \mathsf{Ext}(R; s_1)$ appears uniformly random, ensuring that k and consequently $t = m \oplus k$ are also uniformly random and independent of the message m. This ensures that the adversary gains no advantage, thereby proving the scheme's security.

2.1 Anonymous Incompressible Encryption

SKE schemes. *CCA secure against single encryption.* We initially demonstrate that Dziembowski's incompressible scheme also ensures anonymity. As outlined in the preceding section, in proving the scheme's incompressibility, the final hybrid phase involves generating truly random strings as both the ciphertext and secret key. This absence of any correlation between the ciphertext and secret key inherently establishes anonymity.

Secure against unbounded encryptions. To enhance the scheme against unbounded encryption queries, we employ a programmable pseudorandom function (PPRF). A PPRF is a type of PRF that enables the holder of the PRF key k to perform a specific action - the holder

can produce a programmed key k_{x^*}, such that when the PRF is evaluated using k_{x^*} at x^*, it yields y^* (where the values x^* and y^* are freely chosen by the holder), while maintaining consistency with k at any other point. In Dziembowski's scheme, a ciphertext comprises of (R, t), where $t = m \oplus k$. Instead of generating a one-time key k, we utilize the PPRF to generate fresh keys for each encryption call. Specifically, $k = \mathsf{Ext}(R; s_1) \oplus \mathsf{PPRF.Eval}(s_2, R)$, where s_2 represents a PRF key. To prove security, we leverage the capability to lazily sample s_1 and s_2 by reprogramming s_2 at R^* as necessary where (R^*, t^*) is the challenge ciphertext. One might question whether reprogramming is required at all points R where (R, t) represents the ciphertext generated by the encryption oracle. We demonstrate that this is unnecessary due to the privately programmable security of the PPRF and by not deferring the generation of s_1. For more details, refer to Section 4.2.

Rate-1 secure against single encryption. Moving forward, we proceed to establish that the rate-1 incompressible SKE scheme proposed by Branco *et al.* [BDD22] also ensures anonymity. A ciphertext of this scheme also consists of two part where the first part is encoding of $m \oplus \mathsf{PRG}(s)$ and second part is a one time pad of the PRG seed s. We achieve anonymity by incorporating an additional pseudorandom generator (PRG) to obfuscate the first part of the ciphertext. According to the analysis, this PRG contributes to making the ciphertext seem random even when the secret key is known. For further details, see Section 4.3.

However, unlike in the Dziembowski's scheme, we cannot enhance Branco *et al.*'s scheme to achieve security against unbounded queries using a PPRF. The reason is that the secret key consists of a common reference string (CRS) for a rate-1 incompressible encoding [MW20]. Similar to the proof for the security of Dziembowski's scheme, the deferring of the generation of the secret key to the end of the game is employed in one of the hybrids. In other words, the CRS is inaccessible during the encryption queries. Recall, in our previous scheme, we do not defer the generation of s_1 for this very reason.

PKE scheme. We utilize Branco *et al.*'s transformation that constructs a CPA secure incompressible PKE from incompressible SKE and programmable hash proof system (HPS). To ensure anonymity, we show that we require an anonymous incompressible SKE and an anonymous version of a programmable HPS. Recall that an HPS is associated with a language $\mathcal{L} \subset X$ and $Y \subset X \setminus L$, generated during the setup phase. For our purpose, we define anonymity as follows - for two randomly generated instances of HPS with associated $(\mathcal{L}_0, Y_0)$ and $(\mathcal{L}_1, Y_1)$, an element randomly chosen from Y_0 is computationally indistinguishable from an element randomly chosen from Y_1. A quick examination of the LWE-based programmable HPS construction provided by Branco *et al.* indicates the anonymity of their scheme. In certain stages of our proof, we only require the anonymity property of the SKE scheme. At one crucial step, we rely on the scheme being both anonymous and incompressible simultaneously. Hence, we have formulated our anonymous incompressible security

game that encompass both these features. For the definition of anonymous incompressible security game, refer Definition 12 and for more details on the proof for the security of the anonymous incompressible PKE scheme, see Section 4.4.

Chapter 3

Preliminaries

Throughout this thesis, for any finite set X, $x \leftarrow X$ denotes picking an element x from X uniformly at random. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from the distribution $\mathcal{D}$. For any natural number $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. For any n-bit string $x \in \{0,1\}^n$, we denote x_i as the i^{th} bit of x.

3.1 Randomness Extractors

Definition 1 (Strong Average Min-Entropy Extractor)**.** *A (k, ϵ)-strong average min-entropy extractor is an efficient function $\mathsf{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^m$ such that for all jointly distributed random variable X, Y where X takes values $\{0,1\}^d$ and $\tilde{H}_\infty(X|Y) \geq k$, we have $(U_d, \mathsf{Ext}(X, U_n), Y) \approx_\epsilon (U_d, U_m, Y)$ where U_d, U_n, U_m are uniformly random strings of length d, n, m respectively. Here $H_\infty(X|Y) = -\log \mathop{\mathbb{E}}_{y \leftarrow Y}(\max_x \mathsf{Pr}(X = x|Y = y))$ is the average min-entropy of X conditioned on Y.*

Theorem 2. *There exists an explicit efficient $(k, 2^{-\lambda})$-strong average min-entropy extractor $\mathsf{Ext} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^\lambda$ such that $k = \mathsf{poly}(\lambda)$, $d = \mathsf{poly}(\lambda)$, $n = S + k$ and the depth of the extractor circuit is $\mathsf{poly}(\lambda, \log(n))$.*

We will reply on the following theorem that establishes matrix-vector multiplication is a strong average min-entropy extractor.

Theorem 3 (Leftover Hash Lemma, [DRS04])**.** *Let $n, \ell, \lambda, S \in \mathbb{N}$ such that $\ell = n + 2\log(2^\lambda) + O(1)$. Then, the distribution $(C, Cs, f(s))$ is $1/2^{\lambda+1}$-close to the distribution $(C, t, f(s))$ where $C \leftarrow \{0,1\}^{n \times (\ell+S)}, s \leftarrow \{0,1\}^{\ell+S}, t \leftarrow \{0,1\}^n$ and f is any function from $\{0,1\}^{\ell+S} \to \{0,1\}^S$.*

3.2 Programmable Pseudorandom Functions

A programmable pseudorandom functions $\mathsf{PPRF} = (\mathsf{Setup}, \mathsf{PEval}, \mathsf{Eval})$ with key space $\{\mathcal{K}_\lambda\}_\lambda$, input space $\{\mathcal{X}_\lambda\}_\lambda$ and output space $\{\mathcal{Y}_\lambda\}_\lambda$ consists of the following algorithms.

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm is a randomized algorithm that takes as input the security parameter 1^λ and outputs a master secret key $\mathsf{msk} \in \mathcal{K}_\lambda$.

- $\mathsf{msk}, x, y)$: The program algorithm takes input as the master secret key msk, an input $x \in \mathcal{X}_\lambda$ and an output $y \in \mathcal{Y}_\lambda$. It outputs a secret key $k \in \mathcal{K}_\lambda$.
- $\mathsf{PEval}(k, x)$: The programmed evaluation algorithm is a deterministic algorithm that takes as input a secret key $k \in \mathcal{K}_\lambda$ and $x \in \mathcal{X}_\lambda$ and outputs $y \in \mathcal{Y}_\lambda$.
- $\mathsf{Eval}(\mathsf{msk}, x)$: The evaluation algorithm is a deterministic algorithm that takes as input a master secret key $\mathsf{msk} \in \mathcal{K}_\lambda$ and $x \in \mathcal{X}_\lambda$ and outputs $y \in \mathcal{Y}_\lambda$.

Correctness. A programmable PRF is correct if for all $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$, all inputs $x \in \mathcal{X}_\lambda$, setting $k \leftarrow \mathsf{msk}, x^*, y^*)$, we have

$$\mathsf{PEval}(k, x) = \begin{cases} y^* & \text{if } x = x^* \\ \mathsf{Eval}(\mathsf{msk}, x) & \text{otherwise} \end{cases}$$

Definition 4 (Privacy)**.** *A programmable PRF is private if for all efficient $\mathcal{A}$, the following quantity is negligible:*

$$\mathsf{Adv}^{ppriv}[\mathcal{A}] = |\Pr[\mathsf{Expt}_0^{\mathsf{ppriv}}(\mathcal{A}) = 1] - \Pr[\mathsf{Expt}_1^{\mathsf{ppriv}}(\mathcal{A}) = 1]|$$

where $\mathsf{Expt}_b^{\mathsf{ppriv}}(\mathcal{A})$ is given below:

$\mathsf{Expt}_b^{\mathsf{ppriv}}$ is defined between a challenger and an adversary $\mathcal{A}$ for $\lambda \in \mathbb{N}$, which can make evaluation and challenge queries.

- The challenger obtains $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$ and samples $y^* \leftarrow \mathcal{Y}_\lambda$ uniformly at random.
- The challenger responds to each oracle query type made by $\mathcal{A}$ in the following manner:
  - **Evaluation Oracle:** On input $x \in \mathcal{X}_\lambda$, the challenger returns $y \leftarrow \mathsf{Eval}(\mathsf{msk}, x)$.
  - **Challenge Oracle:** For a pair of inputs $x_0, x_1 \in \mathcal{X}_\lambda$, the challenger returns $k \leftarrow \mathsf{msk}, x_b, y^*)$. Note that $\mathcal{A}$ can make only one query to Challenge Oracle and does not query the evaluation oracle on these points.
- A outputs a bit $b' \in \{0, 1\}$, which is also output by $\mathsf{Expt}_b^{\mathsf{ppriv}}$

Theorem 5 ([PS18])**.** *There exists privately programmable PRF from LWE assumptions.*

3.3 HILL-Entropic Encodings

We recall the notion of HILL-entropic encoding.

Definition 6 (HILL-Entropic Encoding)**.** *An (α, β)-HILL-entropic encoding scheme consists of three PPT algorithms:*

- $\mathsf{Setup}(1^\lambda)$: *The setup algorithm takes the security parameter* 1^λ *as input and outputs a common random string* crs.
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{m})$: *The encoding algorithm takes a common random string* crs *and a message* m *producing an encoding* c.
- $\mathsf{Dec}(\mathsf{crs}, c)$: *The decoding algorithm takes a common random string* crs, *a encoding* c *and produces a message* m.

Correctness. There exists a negligible $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $\mathsf{m} \in \{0,1\}^*$ we have

$$\Pr[\mathsf{Dec}(\mathsf{crs}, \mathsf{Enc}(\mathsf{crs}, \mathsf{m})) = \mathsf{m}] = 1 - \mathrm{negl}(\lambda)$$

α-Expansion. For all $\lambda, k \in \mathbb{N}$ and all $\mathsf{m} \in \{0,1\}^k$ we have

$$|\mathsf{Enc}(\mathsf{crs}, m)| \leq \alpha(\lambda, k)$$

β-Hill-Entropy. There exists an algorithm Sim such that for any polynomial $k = k(\lambda)$ and any ensamble of message $\mathsf{m} = \{\mathsf{m}_\lambda\}$, consider the following "real" experiment:

- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
- $c \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{m}_\lambda)$

and let CRS, C denote the random variables for the corresponding values in the "real" experiment. Also, consider the following "simulated" experiment:

- $(\mathsf{crs}', c') \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{m}_\lambda)$

and let CRS', C' denote the random variables for the corresponding values in the "simulated" experiment. We require that $(CRS, C) \approx_c (CRS', C')$ and $\tilde{H}_\infty(C'|CRS') \geq \beta(\lambda, k)$

We call a (α, β)-HILL-entropic encoding good if $\alpha(\lambda, k) = k(1 + o(1)) + \mathsf{poly}(\lambda)$ and $\beta(\lambda, k) = k(1 - o(1)) - \mathsf{poly}(\lambda)$.

Theorem 7 ([MW20]). *There exists good HILL-entropic encoding from DCR and LWE assumptions in the CRS model.*

3.4 Programmable Hash Proof System

Definition 8 (Y-Programmable Hash Proof System). *A Y-programmable hash proof system is defined over a NP language $\mathcal{L} \subset X$, where each element x in the language $\mathcal{L}$ has a witness w. Additionally there exist a subset $Y \subset X \setminus \mathcal{L}$ and efficient ways to sample a language $\mathcal{L}$ with the corresponding trapdoor $\mathsf{td}_\mathcal{L}$, an $x \in \mathcal{L}$ with its witness w and $x \in Y$ with a corresponding trapdoor td_x.*

- $\mathsf{Gen}(1^\lambda, 1^k)$: *The language generation algorithm takes input as the security parameter* λ *and the encapsulated key size* k*. It outputs public parameters* p *defining a language* $\mathcal{L}$ *and a trapdoor* $\mathsf{td}_\mathcal{L}$ *to that language.*

- $\mathsf{Samp}\mathcal{L}(\mathsf{p})$: *Given the public parameter* p*, it outputs an element* $x \in \mathcal{L}$ *with corresponding witness* w*.*

- $\mathsf{Samp}Y(\mathsf{p}, \mathsf{td}_\mathcal{L})$: *Given the public parameter* p *and a trapdoor* $\mathsf{td}_\mathcal{L}$*, it outputs* $x \in Y$ *and the corresponding trapdoor* td_x*.*

The hash proof system itself consists of these algorithms:

- $\mathsf{KeyGen}(\mathsf{p})$: *The keygen algorithm takes as input the public parameter* p *and outputs a public key* pk *and a secret key* sk*.*

- $\mathsf{Encap}(pk, x, w)$: *The encapsulation algorithm takes input as the public key* pk*, an element* x *and a witness* w*. It outputs an encapsulated key* k*.*

- $\mathsf{Decap}(\mathsf{sk}, x)$: *The decapsulation algorithm takes input as the secret key* sk *and any* $x \in \mathcal{X}$*. It outputs an encapsulated key* k*. Note that* x *can be outside* L*.*

- $\mathsf{td}_L, \mathsf{td}_x, \mathsf{sk}, x, \mathsf{k})$: *The programming algorithm takes as input two trapdoors* $\mathsf{td}_L, \mathsf{td}_x$*, a secret key* sk*, an element* $x \in Y$ *and an encapsulated key* k*. It outputs a new secret key* sk'*.*

Correctness. For all $\lambda, k \in \mathbb{N}$, $(\mathsf{p}, \mathsf{td}_\mathcal{L})$ in the range of $\mathsf{Gen}(1^\lambda, 1^k)$, $(\mathsf{pk}, \mathsf{sk})$ in the range of $\mathsf{KeyGen}(\mathsf{p})$, $x \in \mathcal{L}$ and for $\mathsf{k} \leftarrow \mathsf{Encap}(\mathsf{pk}, x, w)$, we have $\mathsf{k} = \mathsf{Decap}(\mathsf{sk}, x)$ with $|\mathsf{k}| = k$.

Language Indistinguishability. For all $\lambda, k \in \mathbb{N}$, $(\mathsf{p}, \mathsf{td}_\mathcal{L}) \leftarrow \mathsf{Gen}(1^\lambda, 1^k)$, $x \leftarrow \mathsf{Samp}\mathcal{L}(\mathsf{p})$ and $(x^*, \mathsf{td}_{x^*}) \leftarrow \mathsf{Samp}Y(\mathsf{p}, \mathsf{td}_\mathcal{L})$, we have the computational indistinguishability

$$\{\mathsf{p}, x\} \approx_c \{\mathsf{p}, x^*\}$$

Programmability. For all $\lambda, k \in \mathbb{N}$, $(\mathsf{p}, \mathsf{td}_\mathcal{L})$ in the range of $\mathsf{Gen}(1^\lambda, 1^k)$, $(\mathsf{pk}, \mathsf{sk})$ in the range of $\mathsf{KeyGen}(\mathsf{p})$, $\mathsf{k} \in \{0,1\}^k$, (x, td_x) in the range of $\mathsf{Samp}Y(\mathsf{p}, \mathsf{td}_\mathcal{L})$ and $\mathsf{sk}' \leftarrow \mathsf{td}_\mathcal{L}, \mathsf{td}_x, \mathsf{sk}, x, \mathsf{k})$, we have

$$\mathsf{Decap}(\mathsf{sk}', x) = \mathsf{k}$$

Y-programmable Smoothness. For all $\lambda, k \in \mathbb{N}$, $(\mathsf{p}, \mathsf{td}_\mathcal{L})$ in the range of $\mathsf{Gen}(1^\lambda, 1^k)$, $(\mathsf{pk}, \mathsf{sk})$ in the range of $\mathsf{KeyGen}(\mathsf{p})$, $\mathsf{k} \in \{0,1\}^k$, (x, td_x) in the range of $\mathsf{Samp}Y(\mathsf{p}, \mathsf{td}_\mathcal{L})$ and $\mathsf{sk}' \leftarrow \mathsf{td}_\mathcal{L}, \mathsf{td}_x, \mathsf{sk}, x, \mathsf{k})$, we have statistical indistinguishability

$$(\mathsf{pk}, \mathsf{sk}, x) \approx_s (\mathsf{pk}, \mathsf{sk}', x)$$

Anonymity. For all $\lambda, k \in \mathbb{N}$, $\{(\mathsf{p}_i, \mathsf{td}_{\mathcal{L}_i}) \leftarrow \mathsf{Gen}(1^\lambda, 1^k)\}_{i\in\{0,1\}}$ and $\{(x_i^*, \mathsf{td}_{x_i^*}) \leftarrow \mathsf{Samp}Y(\mathsf{p}_i, \mathsf{td}_{\mathcal{L}_i})\}_{i\in\{0,1\}}$, we have the computational indistinguishability

$$\{\mathsf{p}_0, \mathsf{p}_1, x_0^*\} \approx_c \{\mathsf{p}_0, \mathsf{p}_1, x_1^*\}$$

Branco *et al.* [BDD22] provided constructions for programmable hash proof system based on LWE and DDH. In the LWE construction, the public parameter is a matrix $p = A \in \mathbb{Z}_q^{n\times n}$ where the trapdoor is $\mathsf{td}_{\mathcal{L}} = (B, T)$ where B is statistically close to uniform. Therefore, the construction is anonymous.

Theorem 9 ([BDD22]). *There exists anonymous programmable hash proof systems from LWE assumptions.*

3.5 Incompressible Secret Key Encryption

A secret key encryption scheme $\mathsf{SKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\{\mathcal{M}_\lambda\}_\lambda$ consists of the following PPT algorithms.

- $\mathsf{Setup}(1^\lambda, 1^S)$: The setup algorithm is a randomized algorithm that takes as input the security parameter λ and a parameter 1^S and outputs a secret key sk.
- $\mathsf{Enc}(\mathsf{sk}, m)$: The encryption algorithm is a randomized algorithm takes as input a secret key sk and a message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: The decryption algorithm takes as input a secret key sk and a ciphertext ct and outputs either a message $m \in \mathcal{M}_\lambda$ or $\perp$.

Correctness. For correctness, we require that for all $\lambda \in \mathbb{N}, S \in \mathbb{N}, m \in \mathcal{M}_\lambda$ and $\mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda, 1^S)$,

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{sk}, m)) = m] = 1$$

where the probability is over the random bits used in the encryption algorithm.

Incompressible SKE Security (No Queries). Consider the following experiment with an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

- **Initialization Phase:** $\mathcal{A}_1$ on input 1^λ, outputs an upper bound on the state size 1^S. The challenger runs $\mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda, 1^S)$.
- **Challenge Phase:** $\mathcal{A}_1$ outputs two message m_0, m_1, along with an auxiliary information aux. The challenger randomly chooses $b \in \{0, 1\}$ and computes a ciphertext $\mathsf{ct}^* = \mathsf{Enc}(\mathsf{sk}, m_b)$ and sends it to $\mathcal{A}_1$.
- **First Response Phase:** $\mathcal{A}_1$ computes a state st such that $|\mathsf{st}| \leq S$.

- **Second Response Phase:** $\mathcal{A}_2$ receives $(\mathsf{sk}, \mathsf{aux}, \mathsf{st})$ and outputs b'. $\mathcal{A}$ wins the experiment if $b = b'$.

Definition 10. *An SKE scheme is said to be incompressible secure if for all PPT adversaries $\mathcal{A}$, there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}, S \in \mathbb{N}$,

$$\Pr[\mathcal{A} \textit{ wins in the above experiment}] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

The ratio between a message's size and its corresponding ciphertext size is a crucial parameter in any encryption scheme. This ratio is referred to as the *rate* of the scheme. For an efficient scheme, it's essential for this ratio to approach 1.

Definition 11. *Let* $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *be an encryption scheme. Then, the rate of the scheme is defined as* $|m|/|\mathsf{ct}|$ *where* m *is an element from the message space and* $\mathsf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, m)$ *for any* $\mathsf{sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$*. We denote the rate as 1 if*

$$\frac{|m|}{|\mathsf{ct}|} = |m| + o(|m|) \cdot \mathsf{poly}(\lambda)$$

as $|m|$ *tends to infinity.*

Chapter 4

Anonymous Incompressible Encryption schemes

In this chapter, we define anonymous incompressible encryption and present two constructions for anonymous incompressible SKE and one construction for anonymous incompressible PKE. The first SKE construction achieves a rate of $1/2$ but remains secure against unbounded encryption queries. Conversely, the second SKE construction achieves an optimal rate of 1 but is unable to manage encryption queries. The third construction which is PKE handles encryption queries.

4.1 Anonymous Incompressible Enryption: Definition

In this section, we will define the anonymous version of the incompressible security game for SKE setting. Similar to the incompressible SKE schemes, we will consider two adversaries $\mathcal{A}_1, \mathcal{A}_2$. The challenger begins the game by generating two secret keys $(\mathsf{sk}_1, \mathsf{sk}_2)$. In the challenge phase, the first adversary $\mathcal{A}_1$ will send two messages (m_0, m_1). It receives the complete challenge ciphertext which is an encryption of m_b using sk_b and produces a compressed version of the challenge ciphertext. The second adversary $\mathcal{A}_2$ is provided with two secret keys, compressed challenge ciphertext which was created by $\mathcal{A}_1$.

Definition 12. *(Anonymous Incompressible SKE with Unbounded Queries Security). Let* $\mathsf{SKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *be an incompressible SKE scheme. Consider the following experiment with an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

- ***Initialization Phase:*** $\mathcal{A}_1$ *on input* 1^λ*, outputs an upper bound on the state size* 1^S*. The challenger runs* $\{\mathsf{sk}_i\}_{i\in\{0,1\}} \leftarrow \mathsf{Setup}(1^\lambda, 1^S)$.
- ***Pre-Challenge Query Phase:*** $\mathcal{A}_1$ *is allowed to make an unbounded number of queries. For each query* (i, m)
  - *The challenger computes* $\mathsf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_i, m)$ *and returns* ct *to* Adv_1.
- ***Challenge Phase:*** $\mathcal{A}_1$ *outputs a* (m_0, m_1)*, along with an auxiliary information* aux*. The challenger randomly chooses* $b \in \{0, 1\}$ *and computes a ciphertext* $\mathsf{ct}^* = \mathsf{Enc}(\mathsf{sk}_b, m_b)$ *and sends it to* $\mathcal{A}_1$.
- ***Post-Challenge Query Phase:*** $\mathcal{A}_1$ *is allowed to make an unbounded number of queries. For each query* (i, m)
  - *The challenger computes* $\mathsf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_i, m)$ *and returns* ct *to* Adv_1.
- ***First Response Phase:*** $\mathcal{A}_1$ *computes a state* st *such that* $|\mathsf{st}| \leq S$.

- ***Second Response Phase:*** *$\mathcal{A}_2$ receives $(\{\mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{aux}, \mathsf{st})$ and outputs b'. $\mathcal{A}$ wins the experiment if $b = b'$.*

Definition 13. *An SKE scheme is said to be anonymous incompressible with unbounded queries secure if for all PPT adversaries $\mathcal{A}$, there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}, S \in \mathbb{N}$,*

$$\Pr[\mathcal{A} \text{ wins in the above experiment}] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

We also define a version where the first adversary does not have access to the encryption oracle.

Definition 14. *An SKE scheme is said to be anonymous incompressible secure if for all PPT adversaries $\mathcal{A}$, there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}, S \in \mathbb{N}$,*

$$\Pr[\mathcal{A} \text{ wins in the above experiment}] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where $\mathcal{A}_1$ is restricted from making any encryption queries during the pre-challenge and post-challenge query phases.

4.2 Anonymous Incompressible SKE with Unbounded Queries

This scheme closely resembles the incompressible symmetric key encryption scheme proposed by Dziembowski [Dzi06]. Let $\mathsf{PPRF} = (\mathsf{PPRF.Setup}, \mathsf{PPRF.Program}, \mathsf{PPRF.PEval}, \mathsf{PPRF.Eval})$ be a private programmable pseudorandom function with domain $\{0,1\}^{S+\mathsf{poly}(\lambda)}$ and range $\{0,1\}^{\lambda}$, $\mathsf{Ext} : \{0,1\}^{S+\mathsf{poly}(\lambda)} \times \{0,1\}^{\ell} \to \{0,1\}^{\lambda}$ be a strong average min-entropy extractor where $\ell = \mathsf{poly}(\lambda)$, and $\mathsf{PRG} : \{0,1\}^{\lambda} \to \{0,1\}^{S+\mathsf{poly}(\lambda)}$ be a secure pseudorandom generator.

- $\mathsf{Setup}(1^{\lambda}, 1^{S})$: The setup algorithm takes a input the security parameter 1^{λ} and a parameter 1^{S}. It samples $k_1 \leftarrow \{0,1\}^{\ell}$. It generates $\mathsf{msk} \leftarrow \mathsf{PPRF.Setup}(1^{\lambda})$ and samples $r \leftarrow \{0,1\}^{\lambda}$ uniformly at random. It computes $k_2 \leftarrow \mathsf{PPRF.Program}(\mathsf{msk}, \bar{0}, r)$ and sets $\mathsf{sk} = (k_1, k_2)$.
- $\mathsf{Enc}(\mathsf{sk}, m)$: Let $\mathsf{sk} = (k_1, k_2)$. The encryption algorithm takes as input the secret key sk and a message m. It first randomly generates $R \leftarrow \{0,1\}^{S+\mathsf{poly}(\lambda)}$ and computes $s \leftarrow \mathsf{Ext}(R; k_1) \oplus \mathsf{PPRF.PEval}(k_2, R)$. It computes $\mathsf{ske.ct} \leftarrow \mathsf{PRG}(s) \oplus m$ and returns $\mathsf{ct} = (R, \mathsf{ske.ct})$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: Let $\mathsf{sk} = (k_1, k_2)$ and $\mathsf{ct} = (R, \mathsf{ske.ct})$. The decryption algorithm takes as input the secret key sk and a ciphertext ct. It computes $s \leftarrow \mathsf{PPRF.PEval}(k_2, R) \oplus \mathsf{Ext}(R; k_1)$ and returns $m \leftarrow \mathsf{ske.ct} \oplus \mathsf{PRG}(s)$.

Correctness: The correctness of the scheme is straightforward. Let $\mathsf{ct} = (R, \mathsf{ske.ct})$ denote an encryption of m using a secret key $\mathsf{sk} = (k_1, k_2)$. Here, $\mathsf{ske.ct} \leftarrow \mathsf{PRG}(s) \oplus m$, where s

is generated by computing $s \leftarrow \mathsf{Ext}(R; k_1) \oplus \mathsf{PPRF.PEval}(k_2, R)$. The decryption algorithm initiates by computing s in the same manner and then derives m by computing $\mathsf{ske.ct} \oplus \mathsf{PRG}(s)$.

Rate: Let $\mathsf{ct} = (R, \mathsf{ske.ct})$. The ciphertext size is $|R| + |\mathsf{ske.ct}|$. Observe that $|R| = S + \mathsf{poly}(\lambda)$ whereas $|\mathsf{ske.ct}| = |m|$. For the best parameter of S which is $S = |m| - \mathsf{poly}(\lambda)$, we get $|\mathsf{ct}| = 2|m|$. Therefore, the rate is $1/2$.

Theorem 15. *Let* $\mathsf{PPRF} = (\mathsf{PPRF.Setup}, \mathsf{PPRF.Program}, \mathsf{PPRF.PEval}, \mathsf{PPRF.Eval})$ *be a private programmable PRF,* PRG *is a secure PRG and* Ext *be a strong average min-entropy extractor, then the above scheme is an anonymous incompressible SKE scheme secure against unbounded encryption queries.*

Proof. We will show that the above scheme is secure using a series of hybrid arguments.

G_0: This is the incompressible anonymous security game where the challenger randomly chooses $b \in \{0,1\}$ and encrypts one of m_b for using $\mathsf{sk}^{(b)}$ given by $\mathcal{A}_1$.

- **Initialization Phase:**
  1. The challenger randomly generates $\{k_1^{(d)}\}_{d\in\{0,1\}}$.
  2. It generate $\{\mathsf{msk}^{(d)} \leftarrow \mathsf{PPRF.Setup}(1^\lambda)\}_{d\in\{0,1\}}$.
  3. It samples $\{r^{(d)}\}_{d\in\{0,1\}}$ uniformly at random.
  4. It computes $\{k_2^{(d)} \leftarrow \mathsf{PPRF.Program}(\mathsf{msk}^{(d)}, \bar{0}^{(d)}, r^{(d)})\}_{d\in\{0,1\}}$.
  5. It sets $\mathsf{sk}^{(d)} = (k_1^{(d)}, k_2^{(d)})$ for $d \in \{0,1\}$.
- **Pre-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.PEval}(k_2^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. It randomly generates R^*.
  4. It computes $s^* \leftarrow \mathsf{Ext}(R^*, k_1^{(b)}) \oplus \mathsf{PPRF.PEval}(k_2^{(b)}, R^*)$.
  5. It computes $\mathsf{ske.ct}^* \leftarrow m_b \oplus \mathsf{PRG}(s^*)$.
  6. It sends $\mathsf{ct}^* = (R^*, \mathsf{ske.ct}^*)$ to $\mathcal{A}_1$.
- **Post-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.

2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.PEval}(k_2^{(i)}, R)$.
3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
4. It sets $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.

- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**
  1. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to $\mathcal{A}_2$.
  2. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

G_1: The challenger programs the master secret key at R^* instead of $\bar{0}$.

- **Initialization Phase:**
  1. The challenger randomly generates $\{k_1^{(d)}\}_{d\in\{0,1\}}$.
  2. It generate $\{\mathsf{msk}^{(d)} \leftarrow \mathsf{PPRF.Setup}(1^\lambda)\}_{d\in\{0,1\}}$.
  3. It samples $\{r^{(d)}\}_{d\in\{0,1\}}$ uniformly at random.
  4. It randomly generates R^*.
  5. It computes $\{k_2^{(d)} \leftarrow \mathsf{PPRF.Program}(\mathsf{msk}^{(d)}, R^*, r^{(d)})\}_{d\in\{0,1\}}$.
  6. It sets $\mathsf{sk}^{(d)} = (k_1^{(d)}, k_2^{(d)})$ for $d \in \{0,1\}$.
- **Pre-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.PEval}(k_2^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. ~~It randomly generates R^*.~~
  4. It computes $s^* \leftarrow \mathsf{Ext}(R^*, k_1^{(b)}) \oplus \mathsf{PPRF.PEval}(k_2^{(b)}, R^*)$.
  5. It computes $\mathsf{ske.ct}^* \leftarrow m_b \oplus \mathsf{PRG}(s^*)$.
  6. It sends $\mathsf{ct}^* = (R^*, \mathsf{ske.ct}^*)$ to $\mathcal{A}_1$.
- **Post-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.PEval}(k_2^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.

- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**
  1. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to $\mathcal{A}_2$.
  2. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

G_2: The challenger uses $\mathsf{msk}^{(d)}$ instead of $k_2^{(d)}$ in the Query Phases and the programmed keys $k_2^{(d)}$ are generated in the Second Response phase.

- **Initialization Phase:**
  1. The challenger randomly generates $\{k_1^{(d)}\}_{d\in\{0,1\}}$.
  2. It generate $\{\mathsf{msk}^{(d)} \leftarrow \mathsf{PPRF.Setup}(1^\lambda)\}_{d\in\{0,1\}}$.
  3. ~~It samples $\{r^{(d)}\}_{d\in\{0,1\}}$ uniformly at random.~~
  4. It randomly generates R^*.
  5. ~~It computes $\{k_2^{(d)} \leftarrow \mathsf{PPRF.Program}(\mathsf{msk}^{(d)}, R^*, r^{(d)})\}_{d\in\{0,1\}}$.~~
  6. ~~It sets $\mathsf{sk}^{(d)} = (k_1^{(d)}, k_2^{(d)})$ for $d \in \{0,1\}$.~~
- **Pre-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.Eval}(\mathsf{msk}^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. It samples $r^{(b)}$ uniformly at random.
  4. It computes $s^* \leftarrow \mathsf{Ext}(R^*, k_1^{(b)}) \oplus r^{(b)}$.
  5. It computes $\mathsf{ske.ct}^* \leftarrow m_b \oplus \mathsf{PRG}(s^*)$.
  6. It sends $\mathsf{ct}^* = (R^*, \mathsf{ske.ct}^*)$ to $\mathcal{A}_1$.
- **Post-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.Eval}(\mathsf{msk}^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sets $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.

- **Second Response Phase:**
  1. It samples $r^{(1-b)}$ uniformly at random.
  2. It computes $\{k_2^{(d)} \leftarrow \mathsf{PPRF.Program}(\mathsf{msk}^{(d)}, R^*, r^{(d)})\}_{d\in\{0,1\}}$.
  3. It sets $\mathsf{sk}^{(d)} = (k_1^{(d)}, k_2^{(d)})$ for $d \in \{0,1\}$.
  4. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to $\mathcal{A}_2$.
  5. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

G_3: The challenger randomly generates s^* in the Challenge Phase.

- **Initialization Phase:**
  1. The challenger randomly generates $\{k_1^{(d)}\}_{d\in\{0,1\}}$.
  2. It generate $\{\mathsf{msk}^{(d)} \leftarrow \mathsf{PPRF.Setup}(1^\lambda)\}_{d\in\{0,1\}}$.
  3. It randomly generates R^*.
- **Pre-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.Eval}(\mathsf{msk}^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. ~~It samples $r^{(b)}$ uniformly at random.~~
  4. ~~It computes $s^* \leftarrow \mathsf{Ext}(R^*, k_1^{(b)}) \oplus r^{(b)}$.~~
  5. It randomly generates s^*.
  6. It computes $\mathsf{ske.ct}^* \leftarrow m_b \oplus \mathsf{PRG}(s^*)$.
  7. It sends $\mathsf{ct}^* = (R^*, \mathsf{ske.ct}^*)$ to $\mathcal{A}_1$.
- **Post-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.Eval}(\mathsf{msk}^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**

1. It samples $r^{(1-b)}$ uniformly at random.
2. It sets $r^{(b)} = \mathsf{Ext}(R^*; k_1^{(b)}) \oplus s^*$.
3. It computes $\{k_2^{(d)} \leftarrow \mathsf{PPRF.Program}(\mathsf{msk}^{(d)}, R^*, r^{(d)})\}_{d\in\{0,1\}}$.
4. It sets $\mathsf{sk}^{(d)} = (k_1^{(d)}, k_2^{(d)})$ for $d \in \{0,1\}$.
5. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to $\mathcal{A}_2$.
6. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

G_4: The challenger randomly generates $r^{(b)}$ in the Second Response Phase.

- **Initialization Phase:**
  1. The challenger randomly generates $\{k_1^{(d)}\}_{d\in\{0,1\}}$.
  2. It generate $\{\mathsf{msk}^{(d)} \leftarrow \mathsf{PPRF.Setup}(1^\lambda)\}_{d\in\{0,1\}}$.
  3. It randomly generates R^*.
- **Pre-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.Eval}(\mathsf{msk}^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. It randomly generates s^*.
  4. It computes $\mathsf{ske.ct}^* \leftarrow m_b \oplus \mathsf{PRG}(s^*)$.
  5. It sends $\mathsf{ct}^* = (R^*, \mathsf{ske.ct}^*)$ to $\mathcal{A}_1$.
- **Post-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.Eval}(\mathsf{msk}^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $c = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**
  1. It samples $r^{(1-b)}$ uniformly at random.
  2. It randomly generates $r^{(b)}$.
  3. It computes $\{k_2^{(d)} \leftarrow \mathsf{PPRF.Program}(\mathsf{msk}^{(d)}, R^*, r^{(d)})\}_{d\in\{0,1\}}$.

4. It sets $\mathsf{sk}^{(d)} = (k_1^{(d)}, k_2^{(d)})$ for $d \in \{0,1\}$.
5. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to $\mathcal{A}_2$.
6. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

G_5: The challenger randomly generates $\mathsf{ske.ct}^*$ in the Challenge Phase.

- **Initialization Phase:**
  1. The challenger randomly generates $\{k_1^{(d)}\}_{d\in\{0,1\}}$.
  2. It generate $\{\mathsf{msk}^{(d)} \leftarrow \mathsf{PPRF.Setup}(1^\lambda)\}_{d\in\{0,1\}}$.
  3. It randomly generates R^*.
- **Pre-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.Eval}(\mathsf{msk}^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $\mathsf{ct} = (R, \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. ~~It randomly generates s^*.~~
  4. It randomly generates $\mathsf{ske.ct}^*$.
  5. It sends $\mathsf{ct}^* = (R^*, \mathsf{ske.ct}^*)$ to $\mathcal{A}_1$.
- **Post-Challenge Query Phase:** For each query (i, m) from $\mathcal{A}_1$,
  1. The challenger randomly generates R.
  2. It computes $s \leftarrow \mathsf{Ext}(R, k_1^{(i)}) \oplus \mathsf{PPRF.Eval}(\mathsf{msk}^{(i)}, R)$.
  3. It computes $\mathsf{ske.ct} \leftarrow m \oplus \mathsf{PRG}(s)$.
  4. It sends $c = (R \parallel \mathsf{ske.ct})$ to $\mathcal{A}_1$.
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**
  1. It samples $\{r^{(d)}\}_{d\in\{0,1\}}$ uniformly at random.
  2. It computes $\{k_2^{(d)} \leftarrow \mathsf{PPRF.Program}(\mathsf{msk}^{(d)}, R^*, r^{(d)})\}_{d\in\{0,1\}}$.
  3. It sets $\mathsf{sk}^{(d)} = (k_1^{(d)}, k_2^{(d)})$ for $d \in \{0,1\}$.
  4. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to $\mathcal{A}_2$.
  5. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

Analysis: Let $p_{\mathcal{A},i}$ denote the winning probability of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in Game G_i. We will show that the success probability in each game is close to 1/2.

Lemma 16. *From the privacy security of* PPRF*, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},1} - p_{\mathcal{A},0}| \leq \text{negl}(\lambda)$.

Proof. The games G_1 and G_2 are similar, except that the master key is programmed at R^* instead of $\bar{0}$. Therefore, we can use an adversary that distinguishes G_1 to G_2 to build an adversary that break the privacy security of PPRF. Observe that in the entire game, $\mathsf{msk}^{(d)}$ for both $d \in \{0, 1\}$ is only used to generate the programmed keys and not evaluate the PPRF. □

Lemma 17. *From the correctness of* PPRF*, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},2} - p_{\mathcal{A},1}| \leq \text{negl}(\lambda)$.

Proof. The games G_2 and G_3 are similar, except in scenario when R^* is generated in one of the Query phases. If this event doesn't occur, then based on the correctness of PPRF, we have $\mathsf{PPRF.Eval}(\mathsf{msk}^{(i)}, R) = \mathsf{PPRF.PEval}(k_2^{(i)}, R)$ and the fact that $r^{(b)} = \mathsf{PPRF.PEval}(k_2^{(b)}, R^*)$ which is used in the Challenge Phase. The probability of this event occurring is bounded by $q(\lambda)/2^{|R|} = q(\lambda)/2^{S+\mathsf{poly}(\lambda)}$, where q represents the number of queries made by the first adversary. This is because the challenger randomly samples a fresh R for each encryption query. Since the adversary is a probabilistic polynomial-time (PPT) machine, q is polynomial. □

Lemma 18. *For all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},3} - p_{\mathcal{A},2}| = 0$.

Proof. In G_3, $r^{(b)}$ is chosen uniformly at random whereas s^* is set to $\mathsf{Ext}(R^*, k_1^{(b)}) \oplus r^{(b)}$. Whereas, in G_4, s^* is chosen uniformly at random whereas $r^{(b)}$ is set to $\mathsf{Ext}(R^*, k_1^{(b)}) \oplus s^*$. It is easy to see that the two distributions are equivalent. □

Lemma 19. *Assuming that* Ext *is a strong average min-entropy extractor, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},4} - p_{\mathcal{A},3}| = \text{negl}(\lambda)$.

Proof. The difference between G_4 and G_5 lies in generation of $r^{(b)}$. In G_4, it is set to $\mathsf{Ext}(R^*, k_1^{(b)}) \oplus s^*$ whereas in G_5, it is truly random. Note that at-most S bits of information related R is relayed to the second adversary via st. Using this information and the fact that Ext is a strong average min-entropy extractor, we can deduce that

$$(k_1^{(b)}, \mathsf{Ext}(R^*, k_1^{(b)}), \mathsf{st}) \approx_S (k_1^{(b)}, t, \mathsf{st})$$

where t is drawn from the uniform distribution. This implies that the $r^{(b)}$ from the two games are statistically indistinguishable. □

Lemma 20. *Assuming that* PRG *is a secure PRG, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},5} - p_{\mathcal{A},4}| = \text{negl}(\lambda)$.

Proof. Observe that s^* is not used anywhere in G_5 except while compute $\mathsf{ske.ct}^*$. From the fact that PRG is secure, we have $m_b \oplus \mathsf{PRG}(s^*)$ is computationally indistinguishable from a truly random string. Therefore, the two games are indistinguishable. □

Lemma 21. *For all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $p_{\mathcal{A},5} = 1/2$.

Proof. The value b is not used anywhere in the entire G_6 game. Therefore, no adversary can win this game with probability more than $1/2$. □

Using the above lemmas and triangular inequality, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0} \leq 1/2 + \text{negl}(\lambda)$.

□

4.3 Anonymous Incompressible SKE (Rate-1)

Let $\mathsf{IE} = (\mathsf{IE.Setup}, \mathsf{IE.Enc}, \mathsf{IE.Dec})$ be an (α, β)-HILL entropic encoding scheme, $\mathsf{Ext}{:}\{0,1\}^{\alpha(\lambda,n)} \times \{0,1\}^{d(\lambda)} \to \{0,1\}^{2\lambda}$ be a $(\beta(\lambda, n) - S, \text{negl}(\lambda))$ strong average-case min-entropy extractor where $d(\lambda) = \mathsf{poly}(\lambda)$, $\mathsf{PRG}_1 : \{0,1\}^\lambda \to \{0,1\}^n$ and $\mathsf{PRG}_2 : \{0,1\}^\lambda \to \{0,1\}^{\alpha(\lambda,n)}$ be pseudorandom generators.

- $\mathsf{Setup}(1^\lambda, 1^S)$: The setup algorithm takes an input the security parameter 1^λ and a parameter 1^S. It generates $\mathsf{ie.crs} \leftarrow \mathsf{IE.Setup}(1^\lambda)$ and randomly generates $k_1 \leftarrow \{0,1\}^{d(\lambda)}$ and $k_2 \leftarrow \{0,1\}^{2\lambda}$. It sets $\mathsf{sk} = (\mathsf{ie.crs}, k_1, k_2)$.
- $\mathsf{Enc}(\mathsf{sk}, m)$: Let $\mathsf{sk} = (\mathsf{ie.crs}, k_1, k_2)$. The encryption algorithm takes as input the secret key sk and a message m. It first randomly generates s_1, s_2 and computes $\mathsf{ie.cd} \leftarrow \mathsf{IE.Enc}(crs, \mathsf{PRG}_1(s_1) \oplus m)$. It computes $c_0 \leftarrow \mathsf{ie.cd} \oplus \mathsf{PRG}_2(s_2)$ and $c_1 \leftarrow (s_1 \parallel s_2) \oplus \mathsf{Ext}(c_0, k_1) \oplus k_2$. It returns $\mathsf{ct} = (c_0, c_1)$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: Let $\mathsf{sk} = (\mathsf{ie.crs}, k_1, k_2)$ and $\mathsf{ct} = (c_0, c_1)$. The decryption algorithm takes as input the secret key sk and a ciphertext ct. It computes $(s_1 \parallel s_2) \leftarrow c_1 \oplus \mathsf{Ext}(c_0, k_1) \oplus k_2$ and $c' \leftarrow c_0 \oplus \mathsf{PRG}_2(s_2)$. It generates $m \leftarrow \mathsf{IE.Dec}(crs, c') \oplus \mathsf{PRG}_1(s_1)$ and returns m.

Correctness: The correctness of the scheme is straightforward. Let $\mathsf{ct} = (c_0, c_1)$ denote an encryption of m using a secret key $\mathsf{sk} = (\mathsf{ie.crs}, k_1, k_2)$. Here, $c_1 \leftarrow (s_1 \parallel s_2) \oplus \mathsf{Ext}(c_0, k_1) \oplus k_2$ so the decryption algorithm gets $(s_1 \parallel s_2) \leftarrow c_1 \oplus \mathsf{Ext}(c_0, k_1) \oplus k_2$. Also, $c_0 \leftarrow \mathsf{ie.cd} \oplus \mathsf{PRG}_2(s_2)$ so it gets $c' \leftarrow c_0 \oplus \mathsf{PRG}_2(s_2)$ which is equal to ie.cd. Then it gets m by computing $\mathsf{IE.Dec}(\mathsf{crs}, c') \oplus \mathsf{PRG}_1(s_1)$.

Rate: The ciphertexts are of size $2\lambda + \alpha(\lambda, n)$ and key is of size $d(\lambda) + 2\lambda + t(\lambda, n)$ where $t(\lambda, n)$ is the size of the encoding's crs. Notice that the extractor exists if $\beta(\lambda, n) - S - 2\log(\frac{1}{\mathsf{negl}(\lambda)}) + 2 \geq 2\lambda$. So, the adversary is allowed to have a leakage size of $S \leq \beta(\lambda, n) - 2\lambda - 2\log(\frac{1}{\mathsf{negl}(\lambda)}) + 2$. If we choose a good entropic encoding we get a rate of $\frac{n}{n(1+o(1))+\mathsf{poly}(\lambda)}$, allowed leakage of $\mathcal{S} = n(1 - o(1)) - \mathsf{poly}(\lambda)$ and key size of $k = n(1 + o(1)) + \mathsf{poly}(\lambda)$.

Theorem 22. *Let* $\mathsf{IE} = (\mathsf{IE.Setup}, \mathsf{IE.Enc}, \mathsf{IE.Dec})$ *be a HILL entropic encoding scheme,* Ext *be strong average case min entropy extractor,* $\mathsf{PRG}_1, \mathsf{PRG}_2$ *be secure pseudorandom generators, then the above scheme is an anonymous incompressible SKE scheme.*

Proof. We will show that the above scheme is secure using a series of hybrid arguments.

G_0: This is the incompressible anonymous security game where the challenger randomly chooses $b \in \{0, 1\}$ and encrypts one of m_b for using $\mathsf{sk}^{(b)}$ given by $\mathcal{A}_1$.

- **Initialization Phase:**
  1. The challenger generates $\mathsf{ie.crs}^{(d)} \leftarrow \mathsf{IE.Setup}(1^\lambda, 1^S)$
  2. It randomly generates $(k_1^{(d)}, k_2^{(d)})$.
  3. It sets $\mathsf{sk}^{(d)} = (\mathsf{ie.crs}^{(d)}, k_1^{(d)}, k_2^{(d)})$ for $d \in \{0, 1\}$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m0, m1, aux, S) where aux is auxiliary information which will be relayed to the second adversary A2.
  2. It randomly chooses $b \in \{0, 1\}$.
  3. It randomly generates s_1^*, s_2^*.
  4. It computes $\mathsf{ie.cd}^* \leftarrow \mathsf{IE.Enc}(crs^{(b)}, \mathsf{PRG}_1(s_1^*) \oplus m_b)$.
  5. It computes $c_0^* \leftarrow \mathsf{ie.cd}^* \oplus \mathsf{PRG}_2(s_2^*)$.
  6. It computes $c_1^* \leftarrow s^* \oplus \mathsf{Ext}(c_0^*, k_1^{(b)}) \oplus k_2^{(b)}$.
  7. It sends $\mathsf{ct}^* = (c_0^*, c_1^*)$ to $\mathcal{A}_1$.
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**
  1. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$.
  2. $\mathcal{A}_2$ outputs a bit $b' \in \{0, 1\}$.

G_1: The challenger uses Sim instead of $\mathsf{IE.Setup}, \mathsf{IE.Enc}$ in the Challenge Phase.

- **Initialization Phase:**
  1. The challenger generates $\mathsf{ie.crs}^{(1-b)} \leftarrow \mathsf{IE.Setup}(1^\lambda, 1^S)$.
  2. It randomly generates $(k_1^{(d)}, k_2^{(d)})$.

3. It sets $\mathsf{sk}^{(1-b)} = (\mathsf{ie.crs}^{(1-b)}, k_1^{(1-b)}, k_2^{(1-b)})$.

- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. It randomly generates s_1^*, s_2^*.
  4. It computes $(\mathsf{ie.cd}^*, \mathsf{ie.crs}^{(b)}) \leftarrow \mathsf{Sim}(\mathsf{PRG}_1(s_1^*) \oplus m_b)$.
  5. It computes $c_0^* \leftarrow \mathsf{ie.cd}^* \oplus \mathsf{PRG}_2(s_2^*)$.
  6. It computes $c_1^* \leftarrow s^* \oplus \mathsf{Ext}(c_0^*, k_1^{(b)}) \oplus k_2^{(b)}$.
  7. It sends $\mathsf{ct}^* = (c_0^*, c_1^*)$ to $\mathcal{A}_1$.
  8. It sets $\mathsf{sk}^{(b)} = (\mathsf{ie.crs}^{(b)}, k_1^{(b)}, k_2^{(b)})$.
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**
  1. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$.
  2. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

G_2: The challenger sets c_1^* as a truly random string in the Challenge Phase and constructs $\mathsf{sk}^{(b)}$ in the Second Response Phase.

- **Initialization Phase:**
  1. The challenger generates $\mathsf{ie.crs}^{(1-b)} \leftarrow \mathsf{IE.Setup}(1^\lambda, 1^S)$.
  2. It randomly generates $(k_1^{(1-b)}, k_2^{(1-b)})$ and $(k_1^{(b)})$.
  3. It sets $\mathsf{sk}^{(1-b)} = (\mathsf{ie.crs}^{(1-b)}, k_1^{(1-b)}, k_2^{(1-b)})$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. It randomly generates s_1^*, s_2^*.
  4. It computes $(\mathsf{ie.cd}^*, \mathsf{ie.crs}^{(b)}) \leftarrow \mathsf{Sim}(\mathsf{PRG}_1(s_1^*) \oplus m_b)$.
  5. It computes $c_0^* \leftarrow \mathsf{ie.cd}^* \oplus \mathsf{PRG}_2(s_2^*)$.
  6. It randomly generates c_1^*.
  7. It sends $\mathsf{ct}^* = (c_0^*, c_1^*)$ to $\mathcal{A}_1$.
  8. ~~It sets $\mathsf{sk}^{(b)} = (\mathsf{ie.crs}^{(b)}, k_1^{(b)}, k_2^{(b)}, k_3^{(b)})$.~~
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.

- **Second Response Phase:**
  1. The challenger computes $k_2^{(b)} = c_1^* \oplus \mathsf{Ext}(c_0^*, k_1^{(b)}) \oplus s^*$.
  2. It sets $\mathsf{sk}^{(b)} = (\mathsf{ie.crs}^{(b)}, k_1^{(b)}, k_2^{(b)})$.
  3. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$.
  4. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

G_3: The challenger sets $k_2^{(b)}$ to a truly random string in the Second Response Phase.

- **Initialization Phase:**
  1. The challenger generates $\mathsf{ie.crs}^{(1-b)} \leftarrow \mathsf{IE.Setup}(1^\lambda, 1^S)$.
  2. It randomly generates $(k_1^{(1-b)}, k_2^{(1-b)})$ and $(k_1^{(b)})$.
  3. It sets $\mathsf{sk}^{(1-b)} = (\mathsf{ie.crs}^{(1-b)}, k_1^{(1-b)}, k_2^{(1-b)})$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. It randomly generates s_1^*, s_2^*.
  4. It computes $(\mathsf{ie.cd}^*, \mathsf{ie.crs}^{(b)}) \leftarrow \mathsf{Sim}(\mathsf{PRG}_1(s_1^*) \oplus m_b)$.
  5. It computes $c_0^* \leftarrow \mathsf{ie.cd}^* \oplus \mathsf{PRG}_2(s_2^*)$.
  6. It randomly generates c_1^*.
  7. It sends $\mathsf{ct}^* = (c_0^*, c_1^*)$ to $\mathcal{A}_1$.
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**
  1. The challenger computes a truly random string $k_2^{(b)}$.
  2. It sets $\mathsf{sk}^{(b)} = (\mathsf{ie.crs}^{(b)}, k_1^{(b)}, k_2^{(b)})$.
  3. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$.
  4. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

G_4: The challenger changes $\mathsf{PRG}_1(s_1^*) \oplus m_b$ to a truly random string.

- **Initialization Phase:**
  1. The challenger generates $\mathsf{ie.crs}^{(1-b)} \leftarrow \mathsf{IE.Setup}(1^\lambda, 1^S)$.
  2. It randomly generates $(k_1^{(1-b)}, k_2^{(1-b)})$ and $(k_1^{(b)})$.
  3. It sets $\mathsf{sk}^{(1-b)} = (\mathsf{ie.crs}^{(1-b)}, k_1^{(1-b)}, k_2^{(1-b)})$.

- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. It randomly generates s_1^*, s_2^*, t^*.
  4. It computes $(\mathsf{ie.cd}^*, \mathsf{ie.crs}^{(b)}) \leftarrow \mathsf{Sim}(t^*)$.
  5. It computes $c_0^* \leftarrow \mathsf{ie.cd}^* \oplus \mathsf{PRG}_2(s_2^*)$.
  6. It randomly generates c_1^*.
  7. It sends $\mathsf{ct}^* = (c_0^*, c_1^*)$ to $\mathcal{A}_1$.
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**
  1. The challenger computes a truly random string $k_2^{(b)}$.
  2. It sets $\mathsf{sk}^{(b)} = (\mathsf{ie.crs}^{(b)}, k_1^{(b)}, k_2^{(b)})$.
  3. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$.
  4. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

G_5: The challenger random generates c_0^*.

- **Initialization Phase:**
  1. The challenger generates $\mathsf{ie.crs}^{(1-b)} \leftarrow \mathsf{IE.Setup}(1^\lambda, 1^S)$.
  2. It randomly generates $(k_1^{(1-b)}, k_2^{(1-b)})$ and $(k_1^{(b)})$.
  3. It sets $\mathsf{sk}^{(1-b)} = (\mathsf{ie.crs}^{(1-b)}, k_1^{(1-b)}, k_2^{(1-b)})$.
- **Challenge Phase:**
  1. The first adversary $\mathcal{A}_1$ sends (m_0, m_1, aux, S) where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  2. It randomly chooses $b \in \{0,1\}$.
  3. It randomly generates s_1^*, s_2^*, t^*.
  4. It computes $(\mathsf{ie.cd}^*, \mathsf{ie.crs}^{(b)}) \leftarrow \mathsf{Sim}(t^*)$.
  5. It randomly generates c_0^*.
  6. It randomly generates c_1^*.
  7. It sends $\mathsf{ct}^* = (c_0^*, c_1^*)$ to $\mathcal{A}_1$.
- **First Response Phase:** $\mathcal{A}_1$ outputs a state st such that $|\mathsf{st}| \leq S$.
- **Second Response Phase:**
  1. The challenger computes a truly random string $k_2^{(b)}$.
  2. It sets $\mathsf{sk}^{(b)} = (\mathsf{ie.crs}^{(b)}, k_1^{(b)}, k_2^{(b)})$.
  3. The challenger sends $(\left\{\mathsf{sk}^{(d)}\right\}_{d\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$.
  4. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$.

Analysis: Let $p_{\mathcal{A},i}$ denote the winning probability of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in Game G_i. We will show that the success probability in each game is close to 1/2.

Lemma 23. *Assuming that* IE *is a secure HILL entropic encoding scheme, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},1} - p_{\mathcal{A},0}| \leq \text{negl}(\lambda)$.

Proof. Instead of sampling $\mathsf{ie.crs}^b$ for the entropic encoding using $\mathsf{IE.Setup}$ and then encoding $\mathsf{PRG}_1(s_1^*) \oplus m_b$ in G_1, we simulate both the steps using Sim. If $\mathcal{A}$ can distinguish G_1 from G_2 then it can distinguish $(\mathsf{ie.crs}^b, \mathsf{ie.cd}^*)$ of G_0 from $(\mathsf{ie.crs}^b, \mathsf{ie.cd}^*)$ of G_2 which can be used to create an adversary $\mathcal{A}'$ to break Hill entropic encoding scheme. □

Lemma 24. *For all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},2} - p_{\mathcal{A},1}| \leq \text{negl}(\lambda)$.

Proof. In G_2, $k_2^{(b)}$ is chosen uniformly at random and then c_1^* is set to $s^* \oplus \mathsf{Ext}(c_0^*, k_1^{(b)}) \oplus k_2^{(b)}$. Whereas, in G_3, c_1^* is chosen uniformly at random and $k_2^{(b)}$ is set to $c_1^* \oplus \mathsf{Ext}(c_0^*, k_1^{(b)}) \oplus s^*$. It is easy to see that the two distributions are equivalent. □

Lemma 25. *Assuming that* Ext *is a strong average min-entropy extractor, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},3} - p_{\mathcal{A},2}| \leq \text{negl}(\lambda)$.

Proof. The difference between G_3 and G_4 lies in the generation of $k_2^{(b)}$. In G_3, it is set to $c_1^* \oplus \mathsf{Ext}(c_0^*, k_1^{(b)}) \oplus s^*$ whereas in G_4, it is truly random. Let $C_0^*, C_1^*, CRS, K_1^{(b)}, K_2^{(b)}$ and ST denote the random variable for the corresponding values in the experiment and U_λ independent uniform randomness of length λ. By the $\beta-$Hill entropy of the entropic encoding, we know that $H_\infty(C_0^*|CRS) \geq \beta$. Using the fact that for random variables X, Y, Z, where Y is supported over a set of size T, $H_\infty(X|(Y,Z)) \geq H_\infty((X,Y)|Z) - \log(T) \geq H_\infty(X|Z) - \log(T)$, we deduce that $H_\infty(C_0^*|CRS, K_2^{(b)}, ST, C_1^*) \geq \beta - 2\lambda - log(S)$. Therefore, the extractor gives us that $(K_1^{(b)}, K_2^{(b)}, CRS, ST, U_\lambda)$ is statistically close to $(K_1^{(b)}, K_2^{(b)}, CRS, ST, \mathsf{Ext}(C_0^*, K_1^{(b)}))$. □

Lemma 26. *Assuming that* PRG *is a secure PRG, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},4} - p_{\mathcal{A},3}| \leq \text{negl}(\lambda)$.

Proof. As s_1^* is not used anywhere except for generating h in G_4. So using the fact that PRG is secure, we have $\mathsf{ie.cd}^* \oplus \mathsf{PRG}(s_2^*)$ is computationally indistinguishable from a truly random string. Therefore, the two games are indistinguishable. □

Lemma 27. *Assuming that* PRG *is a secure PRG, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},5} - p_{\mathcal{A},4}| \leq \text{negl}(\lambda)$.

Proof. As s_2^* is not used anywhere except for in the Sim in G_5. So using the fact that PRG is secure, we have $\mathsf{PRG}(s_1^*) \oplus m_b$ is computationally indistinguishable from a truly random string. Therefore, the two games are indistinguishable. □

Lemma 28. *For all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function* negl *such that $p_{\mathcal{A},5} = 1/2 + \mathrm{negl}(\lambda)$.*

Proof. The value b is not used anywhere in the entire G_6 game except for b being revealed through sk in the Second Response Phase. For $d = b$, the crs is generated using Sim whereas for $d = 1 - b$, it is generated using Setup. Now, using HILL entropic encoding scheme we can say that crs^{1-b} generated using the $\mathsf{Setup}(1^\lambda)$ is indistinguishable if it is generated using $\mathsf{Sim}(t^{(1-b)})$ where $t^{(1-b)}$ is generated randomly. So, we can remove the information of b which is being revealed in this way. Therefore, no adversary can win this game with a probability significantly greater than 1/2. □

Using the above lemmas and triangular inequality, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0} \leq 1/2 + \mathrm{negl}(\lambda)$.

□

4.4 Incompressible Anonymous CPA PKE (Rate-1)

Let $\mathsf{HPS} = (\mathsf{KeyGen}', \mathsf{Encap}', \mathsf{Decap}',)$ be a programmable hash proof system with encapsulated keys of size $k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)$, and $\mathsf{SKE} = (\mathsf{Setup}_{\mathsf{AnonSym}}, \mathsf{Enc}_{\mathsf{AnonSym}}, \mathsf{Dec}_{\mathsf{AnonSym}})$ be an anonymous incompressible secure SKE where Setup generates truly random string as secret, the message size is n, keys of size $k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)$ and ciphertexts of size $l(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)$ with the anonymous incompressible SKE adversary being allowed to leak a state of size $\mathcal{S}_{\mathsf{AnonSym}}$.

Construction:

- $\mathsf{Setup}(1^\lambda, 1^\mathcal{S})$:
  - *Generate language and corresponding trapdoor* $(\mathsf{p}, \mathsf{td}_\mathcal{L}) \leftarrow \mathsf{Gen}(1^\lambda, 1^\mathcal{S})$
  - *Return* p and $\mathsf{td}_\mathcal{L}$
- $\mathsf{KeyGen}(\mathsf{p})$:
  - *Let* $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$
  - *Return* $\mathsf{pk} = \mathsf{pk}'$ *and* $\mathsf{sk} = \mathsf{sk}'$
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{m})$:
  - *Parse* $\mathsf{pk} = \mathsf{pk}'$
  - *Let* $(x, w) \leftarrow \mathsf{Samp}\mathcal{L}(p)$

- *Let* $\mathsf{k} \leftarrow \mathsf{Encap}'(\mathsf{pk}', x, w)$
- *Let* $c \leftarrow \mathsf{Enc}_{AnonSym}(\mathsf{k}, \mathsf{m})$
- *Return* $\mathsf{ct} = (x, c)$

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$:
  - *Parse* $\mathsf{sk} = \mathsf{sk}'$
  - *Parse* $\mathsf{ct} = (x, c)$
  - *Let* $\mathsf{k} \leftarrow \mathsf{Decap}'(\mathsf{sk}', x)$
  - *Return* $\mathsf{m} = \mathsf{Dec}_{AnonSym}(\mathsf{k}, c)$

Parameters. The scheme is similar to Branco *et al.* [BDD22] with except that the underlying rate-1 incompressible scheme is anonymous. Therefore, the rate is 1.

Correctness. The correctness of the scheme is straightforward. Let $\mathsf{ct} = (x, c)$ denote an encryption of m using the public key $\mathsf{pk} = \mathsf{pk}'$. The secret key $\mathsf{sk} = \mathsf{sk}'$. From the correctness of the Y-programmable hash proof system, we have $\mathsf{Encap}'(pk', x, w) = \mathsf{Decap}'(sk', x)$ so the decryption algorithm gets k. Now, it gets the message m as $\mathsf{Dec}_{\mathsf{AnonSym}}(k, c)$ which will give the correct m using the correctness of anonymous incompressible SKE.

Theorem 29 (Security)**.** *Let* $(\mathsf{KeyGen}', \mathsf{Encap}', \mathsf{Decap}',)$ *is an anonymous programmable hash proof system and* $\mathsf{SKE} = (\mathsf{Setup}_{\mathsf{AnonSym}}, \mathsf{Enc}_{\mathsf{AnonSym}}, \mathsf{Dec}_{\mathsf{AnonSym}})$ *is an anonymous incompressible secure* SKE*, then the above* PKE *scheme is a secure anonymous incompressible* CPA PKE *scheme.*

Proof. We prove the security by a hybrid argument. We first list all the hybrids and then argue their indistinguishability. In each hybrid, we highlight the changes compared to the previous one.

G_0: This is the anonymous incompressible CPA security game where the challenger randomly chooses $b = 0$ and encrypts the message m_0 given by $\mathcal{A}_1$ using pk_0.

- **First Phase:**
  1. The challenger runs $\mathsf{Setup}(1^\lambda, 1^\mathcal{S})$ to get $\{(\mathsf{pk}_d, \mathsf{sk}_d)\}_{d\in\{0,1\}}$
  2. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  3. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  4. The challenger runs $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \mathsf{m}_0)$ to encrypt m using pk_0
  5. It sends ct to $\mathcal{A}_1$
  6. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$
- **Second Phase:**

1. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
2. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$

G_1: We explicitly represent what happens in KeyGen and Enc.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^\mathcal{S})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_\mathcal{L}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0,1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0,1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_0, w_0) \leftarrow \mathsf{Samp}\mathcal{L}(\mathsf{p}^{(0)})$
  7. It computes $\mathsf{k}_0 \leftarrow \mathsf{Encap}'(\mathsf{pk}'_0, x_0, w_0)$
  8. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}_0, \mathsf{m}_0)$
  9. Let $\mathsf{ct} = (x_0, c)$
  10. It sends ct to $\mathcal{A}_1$
  11. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$
- **Second Phase:**
  1. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  2. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$

G_2: The challenger uses the decapsulation mechanism to encrypt the challenge message instead of encapsulation.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^\mathcal{S})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_\mathcal{L}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0,1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0,1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_0, w_0) \leftarrow \mathsf{Samp}\mathcal{L}(\mathsf{p}^{(0)})$
  7. It computes $\mathsf{k}_0 \leftarrow \mathsf{Decap}'(\mathsf{sk}'_0, x_0)$

8. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}_0, \mathsf{m}_0)$
9. Let $\mathsf{ct} = (x_0, c)$
10. It sends ct to $\mathcal{A}_1$
11. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$

- **Second Phase:**
  1. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  2. $\mathcal{A}_2$ outputs a bit $b' \in \{0, 1\}$

G_3: The challenger samples x_0 from $Y \subset X \setminus \mathcal{L}$ instead of $\mathcal{L}$

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^\mathcal{S})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_\mathcal{L}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0, 1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0, 1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_0, \mathsf{td}_{x_0}) \leftarrow \mathsf{Samp}Y(\mathsf{p}^{(0)}, \mathsf{td}_\mathcal{L}^{(0)})$
  7. It computes $\mathsf{k}_0 \leftarrow \mathsf{Decap}'(\mathsf{sk}'_0, x_0)$
  8. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}_0, \mathsf{m}_0)$
  9. Let $\mathsf{ct} = (x_0, c)$
  10. It sends ct to $\mathcal{A}_1$
  11. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$

- **Second Phase:**
  1. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  2. $\mathcal{A}_2$ outputs a bit $b' \in \{0, 1\}$

G_4: The challenger programs the secret key sk_0 given to the adversary to decapsulate x_0 to the randomly chosen key k.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^\mathcal{S})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_\mathcal{L}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0, 1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0, 1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$

5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
6. The challenger computes $(x_0, \mathsf{td}_{x_0}) \leftarrow \mathsf{Samp}Y(\mathsf{p}^{(0)}, \mathsf{td}_{\mathcal{L}}^{(0)})$
7. It computes $\mathsf{k}_0 \leftarrow \{0,1\}^{k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)}$
8. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}_0, \mathsf{m}_0)$
9. Let $\mathsf{ct} = (x_0, c)$
10. It sends ct to $\mathcal{A}_1$
11. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$

- **Second Phase:**
  1. The challenger computes $\mathsf{sk}_0' \leftarrow (\mathsf{td}_{\mathcal{L}}, \mathsf{td}_{x_0}, \mathsf{sk}_0, x_0, \mathsf{k}_0)$ and set $\mathsf{sk}_0 = \mathsf{sk}_0'$
  2. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  3. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$

G_5: The challenger uses a random k to encryption m_1 as the challenge.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^{\mathcal{S}})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_{\mathcal{L}}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}_d', \mathsf{sk}_d') \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0,1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}_d')$ and $\mathsf{sk}_d = \mathsf{sk}_d'$ for $d \in \{0,1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_0, \mathsf{td}_{x_0}) \leftarrow \mathsf{Samp}Y(\mathsf{p}^{(0)}, \mathsf{td}_{\mathcal{L}}^{(0)})$
  7. It computes $\mathsf{k}_0 \leftarrow \{0,1\}^{k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)}$
  8. It computes $\mathsf{k} \leftarrow \{0,1\}^{k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)}$
  9. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}, \mathsf{m}_1)$
  10. Let $\mathsf{ct} = (x_0, c)$
  11. It sends ct to $\mathcal{A}_1$
  12. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$

- **Second Phase:**
  1. The challenger computes $\mathsf{sk}_0' \leftarrow (\mathsf{td}_{\mathcal{L}}, \mathsf{td}_{x_0}, \mathsf{sk}_0, x_0, \mathsf{k}_0)$ and set $\mathsf{sk}_0 = \mathsf{sk}_0'$
  2. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  3. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$

G_6: The challenger uses decapsulation mechanism to generate k_0.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^{\mathcal{S}})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_{\mathcal{L}}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0,1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0,1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_0, \mathsf{td}_{x_0}) \leftarrow \mathsf{Samp}Y(\mathsf{p}^{(0)}, \mathsf{td}_{\mathcal{L}}^{(0)})$
  7. It computes $\mathsf{k}_0 \leftarrow \mathsf{Decap}'(\mathsf{sk}_0, x_0)$
  8. It computes $\mathsf{k} \leftarrow \{0,1\}^{k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)}$
  9. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}, \mathsf{m}_1)$
  10. Let $\mathsf{ct} = (x_0, c)$
  11. It sends ct to $\mathcal{A}_1$
  12. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$
- **Second Phase:**
  1. ~~The challenger computes $\mathsf{sk}'_0 \leftarrow (\mathsf{td}_{\mathcal{L}}, \mathsf{td}_{x_0}, \mathsf{sk}_0, x_0, \mathsf{k}_0)$ and set $\mathsf{sk}_0 = \mathsf{sk}'_0$~~
  2. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  3. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$

G_7: The challenger generates x_1 using $\mathsf{p}^{(1)}$ instead of x_0 using $\mathsf{p}^{(0)}$.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^{\mathcal{S}})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_{\mathcal{L}}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0,1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0,1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_1, \mathsf{td}_{x_1}) \leftarrow \mathsf{Samp}Y(\mathsf{p}^{(1)}, \mathsf{td}_{\mathcal{L}}^{(1)})$
  7. It computes $\mathsf{k}_1 \leftarrow \mathsf{Decap}'(\mathsf{sk}_1, x_1)$
  8. It computes $\mathsf{k} \leftarrow \{0,1\}^{k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)}$
  9. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}, \mathsf{m}_1)$
  10. Let $\mathsf{ct} = (x_1, c)$
  11. It sends ct to $\mathcal{A}_1$
  12. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$
- **Second Phase:**

1. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
2. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$

G_8: The challenger programs sk_1 at x_1.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^\mathcal{S})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_\mathcal{L}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0,1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0,1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_1, \mathsf{td}_{x_1}) \leftarrow \mathsf{Samp}Y(\mathsf{p}^{(1)}, \mathsf{td}_\mathcal{L}^{(1)})$
  7. It computes $\mathsf{k}_1 \leftarrow \{0,1\}^{k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)}$
  8. It computes $\mathsf{k} \leftarrow \{0,1\}^{k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)}$
  9. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}, \mathsf{m}_1)$
  10. Let $\mathsf{ct} = (x_1, c)$
  11. It sends ct to $\mathcal{A}_1$
  12. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$
- **Second Phase:**
  1. Let $\mathsf{sk}'_1 \leftarrow (\mathsf{td}_\mathcal{L}, \mathsf{td}_{x_1}, \mathsf{sk}'_1, x_1, \mathsf{k}_1)$ and set $\mathsf{sk}_1 = \mathsf{sk}'_1$
  2. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  3. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$

G_9: The challenger uses k_1 to encrypt m_1.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^\mathcal{S})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_\mathcal{L}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0,1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0,1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_1, \mathsf{td}_{x_1}) \leftarrow \mathsf{Samp}Y(\mathsf{p}^{(1)}, \mathsf{td}_\mathcal{L}^{(1)})$
  7. It computes $\mathsf{k}_1 \leftarrow \{0,1\}^{k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)}$
  8. ~~It computes $\mathsf{k} \leftarrow \{0,1\}^{k(\lambda, \mathcal{S}_{\mathsf{AnonSym}}, n)}$~~

9. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}_1, \mathsf{m}_1)$
10. Let $\mathsf{ct} = (x_1, c)$
11. It sends ct to $\mathcal{A}_1$
12. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$

- **Second Phase:**
  1. Let $\mathsf{sk}'_1 \leftarrow (\mathsf{td}_{\mathcal{L}}, \mathsf{td}_{x_1}, \mathsf{sk}'_1, x_1, \mathsf{k}_1)$ and set $\mathsf{sk}_1 = \mathsf{sk}'_1$
  2. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  3. $\mathcal{A}_2$ outputs a bit $b' \in \{0, 1\}$

G_{10}: The challenger uses decapsulation mechanism to generate k_1.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^{\mathcal{S}})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_{\mathcal{L}}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0, 1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0, 1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_1, \mathsf{td}_{x_1}) \leftarrow \mathsf{Samp}Y(\mathsf{p}^{(1)}, \mathsf{td}_{\mathcal{L}}^{(1)})$
  7. It computes $\mathsf{k}_1 \leftarrow \mathsf{Decap}'(\mathsf{sk}_1, x_1)$
  8. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}_1, \mathsf{m}_1)$
  9. Let $\mathsf{ct} = (x_1, c)$
  10. It sends ct to $\mathcal{A}_1$
  11. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$

- **Second Phase:**
  1. ~~Let $\mathsf{sk}'_1 \leftarrow (\mathsf{td}_{\mathcal{L}}, \mathsf{td}_{x_1}, \mathsf{sk}'_1, x_1, \mathsf{k}_1)$ and set $\mathsf{sk}_1 = \mathsf{sk}'_1$~~
  2. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  3. $\mathcal{A}_2$ outputs a bit $b' \in \{0, 1\}$

G_{11}: The challenger generates x_1 from the language.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^{\mathcal{S}})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_{\mathcal{L}}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0, 1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0, 1\}$

4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
6. The challenger computes $(x_1, w_1) \leftarrow \mathsf{Samp}\mathcal{L}(\mathsf{p}^{(1)})$
7. It computes $\mathsf{k}_1 \leftarrow \mathsf{Decap}'(\mathsf{sk}_1, x_1)$
8. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}_1, \mathsf{m}_1)$
9. Let $\mathsf{ct} = (x_1, c)$
10. It sends ct to $\mathcal{A}_1$
11. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$

- **Second Phase:**
  1. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  2. $\mathcal{A}_2$ outputs a bit $b' \in \{0, 1\}$

G_{12}: The challenger uses the encapsulation mechanism to generate k_1.

- **First Phase:**
  1. The challenger runs $\mathsf{Gen}(1^\lambda, 1^\mathcal{S})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_\mathcal{L}^{(d)})\}_{d\in\{0,1\}}$
  2. It computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0, 1\}$
  3. It sets $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0, 1\}$
  4. It sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  5. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  6. The challenger computes $(x_1, w_1) \leftarrow \mathsf{Samp}\mathcal{L}(\mathsf{p}^{(1)})$
  7. It computes $\mathsf{k}_1 \leftarrow \mathsf{Encap}'(\mathsf{pk}_1, x_1, w_1)$
  8. It computes $c \leftarrow \mathsf{Enc}_{\mathsf{AnonSym}}(\mathsf{k}_1, \mathsf{m}_1)$
  9. Let $\mathsf{ct} = (x_1, c)$
  10. It sends ct to $\mathcal{A}_1$
  11. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$

- **Second Phase:**
  1. The challenger sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  2. $\mathcal{A}_2$ outputs a bit $b' \in \{0, 1\}$

Analysis: Let $p_{\mathcal{A},i}$ denote the probability of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ outputting 0 in Game G_i. We will show that the probability in each game is close to 1/2.

Lemma 30. *For all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, $|p_{\mathcal{A},1} - p_{\mathcal{A},0}| = 0$.

Proof. The difference between H_0 and H_1 are purely syntactical. In H_1 we just show more details of what's going inside of KeyGen and Enc. It is easy to see that the two games are equivalent. □

Lemma 31. *From the correctness of programmable* HPS*, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},2} - p_{\mathcal{A},1}| \leq \text{negl}(\lambda)$.

Proof. The game G_1 and G_2 are similar except for the k_0 is computed using Decap' in G_1 whereas it is computed using Encap' in G_0. Using the correctness we can easily see that $\mathsf{Encap}'(\mathsf{pk}'_0, x_0, w_0) = \mathsf{Decap}'(\mathsf{sk}'_0, x_0)$. So, these two games are indistinguishable. □

Lemma 32. *From the language indistinguishability of programmable* HPS*, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},3} - p_{\mathcal{A},2}| \leq \text{negl}(\lambda)$.

Proof. The game G_2 and G_3 are similar except for the x_0 is sampled from $Y \subset X \setminus \mathcal{L}$ in G_3 instead of $\mathcal{L}$. Using the language indistinguishability of the programmable HPS, we can see that the two games are computationally indistinguishable. □

Lemma 33. *From the* Y*-programmable smoothness of programmable* HPS*, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},4} - p_{\mathcal{A},3}| \leq \text{negl}(\lambda)$.

Proof. The game G_3 and G_4 are similar except for the sk'_0 is programmed in G_4 with sampling k_0 randomly. According to the programmable smoothness of hash proof system if $x_0 \notin \mathcal{L}$ and k_0 is chosen randomly then

$$(\mathsf{pk}'_0, \mathsf{sk}'_0, x_0) \overset{\text{s}}{\approx} (\mathsf{pk}'_0, (\mathsf{td}_{\mathcal{L}}, \mathsf{td}_{x_0}, \mathsf{sk}'_0, x_0, \mathsf{k}_0), x_0)$$

Hence, the two games will be statistically close. □

Lemma 34. *Assuming anonymous incompressible security of* $\mathsf{PKE}_{\mathsf{AnonSym}}$*, for all PPT adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl *such that* $|p_{\mathcal{A},5} - p_{\mathcal{A},4}| \leq \text{negl}(\lambda)$.

Proof. The game G_4 and G_5 are similar except that c is an encryption of m_0 using k_0 in G_4 and in G_5, it is an encryption of m_1 using k. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary that can distinguish G_5 from G_4. We will construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that breaks anonymous incompressible security of the underlying SKE scheme.

- **First Phase:**
  1. The challenger generates k_0, k randomly.
  2. $\mathcal{B}_1$ runs $\mathsf{Gen}(1^\lambda, 1^\mathcal{S})$ to get $\{(\mathsf{p}^{(d)}, \mathsf{td}_\mathcal{L}^{(d)})\}_{d\in\{0,1\}}$
  3. $\mathcal{B}_1$ computes $(\mathsf{pk}'_d, \mathsf{sk}'_d) \leftarrow \mathsf{KeyGen}'(\mathsf{p}^{(d)})$ for $d \in \{0,1\}$
  4. $\mathcal{B}_1$ set $\mathsf{pk}_d = (\mathsf{p}^{(d)}, \mathsf{pk}'_d)$ and $\mathsf{sk}_d = \mathsf{sk}'_d$ for $d \in \{0,1\}$
  5. $\mathcal{B}_1$ sends $(\mathsf{pk}_0, \mathsf{pk}_1)$ to $\mathcal{A}_1$
  6. $\mathcal{A}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux)$ where aux is auxiliary information which will be relayed to the second adversary $\mathcal{A}_2$.
  7. $\mathcal{B}_1$ computes $(x_0, \mathsf{td}_{x_0}) \leftarrow \mathsf{Samp}Y(\mathsf{p}^{(0)}, \mathsf{td}_\mathcal{L}^{(0)})$
  8. $\mathcal{B}_1$ sends $(\mathsf{m}_0, \mathsf{m}_1, aux')$ where aux' contains $\mathcal{B}_1$ entire state information.
  9. The challenger sends c to $\mathcal{B}_1$ which is either an encryption of m_0 using k_0 or m_1 using k.
  10. Let $\mathsf{ct} = (x_0, c)$
  11. $\mathcal{B}_1$ sends ct to $\mathcal{A}_1$
  12. $\mathcal{A}_1$ sends a state st such that $|\mathsf{st}| \leq \mathcal{S}$ to $\mathcal{B}_1$ who sends it to the challenger.
- **Second Phase:**
  1. The challenger sends $(k_0, k, \mathsf{st}, aux')$ to $\mathcal{B}_2$ where aux' contains all the necessary trapdoors.
  2. $\mathcal{B}_2$ computes $\mathsf{sk}'_0 \leftarrow (\mathsf{td}_\mathcal{L}, \mathsf{td}_{x_0}, \mathsf{sk}_0, x_0, \mathsf{k}_0)$ and set $\mathsf{sk}_0 = \mathsf{sk}'_0$.
  3. $\mathcal{B}_2$ sends $(\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{0,1\}}, \mathsf{st}, aux)$ to the second adversary $\mathcal{A}_2$
  4. $\mathcal{A}_2$ outputs a bit $b' \in \{0,1\}$

Observe that if c is an encryption of m_0 using k_0, then $\mathcal{B}$ has simulated G_4. Else, it has simulated G_5. □

Lemma 35. *From the Y-programmable smoothness of programmable* HPS*, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function* negl *such that $|p_{\mathcal{A},6} - p_{\mathcal{A},5}| \leq \mathrm{negl}(\lambda)$.*

Proof. The proof is similar to Lemma 33. □

Lemma 36. *Assuming anonymity of* HPS*, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function* negl *such that $|p_{\mathcal{A},7} - p_{\mathcal{A},6}| \leq \mathrm{negl}(\lambda)$.*

Proof. The game G_7 and G_8 are similar except for x_0 is used in G_7 which is an element from the first language and x_2 is used in G_8 which is an element from the second language. According to the anonymity property of hash proof system, we have

$$\{\mathsf{p}^{(0)}, \mathsf{p}^{(1)}, x_0^*\} \approx_c \{\mathsf{p}^{(0)}, \mathsf{p}^{(1)}, x_1^*\}$$

Hence, the two games will be computationally close. □

Lemma 37. *From the Y-programmable smoothness of programmable* HPS*, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function* negl *such that $|p_{\mathcal{A},8} - p_{\mathcal{A},7}| \leq \text{negl}(\lambda)$.*

Proof. The proof is similar to Lemma 33. □

Lemma 38. *Assuming anonymous incompressible security of $\mathsf{PKE}_{\mathsf{AnonSym}}$, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function* negl *such that $|p_{\mathcal{A},9} - p_{\mathcal{A},8}| \leq \text{negl}(\lambda)$.*

Proof. The proof is similar to Lemma 34 except $\mathcal{B}_1$ sends (m_1, m_1, aux') to its challenger instead of (m_0, m_1, aux'). □

Lemma 39. *From the Y-programmable smoothness of programmable* HPS*, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function* negl *such that $|p_{\mathcal{A},10} - p_{\mathcal{A},9}| \leq \text{negl}(\lambda)$.*

Proof. The proof is similar to Lemma 33. □

Lemma 40. *From the language indistinguishability of programmable* HPS*, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function* negl *such that $|p_{\mathcal{A},11} - p_{\mathcal{A},10}| \leq \text{negl}(\lambda)$.*

Proof. The proof is similar to Lemma 32. □

Lemma 41. *From the correctness of programmable* HPS*, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function* negl *such that $|p_{\mathcal{A},12} - p_{\mathcal{A},11}| \leq \text{negl}(\lambda)$.*

Proof. The proof is similar to Lemma 31. □

Observe that Game 12 is the original anonymous incompressible PKE game with the challenge bit $b = 1$. Using the above lemmas and triangular inequality, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, $|p_{\mathcal{A},0} - p_{\mathcal{A},12}| \leq \text{negl}(\lambda)$.

□

Chapter 5

Conclusion

In this thesis, we explored advanced cryptographic concepts, focusing on incompressible encryption and anonymity. We examined security models such as CPA and CCA to frame our discussions, starting from the fundamentals of SKE and PKE. Our key contributions include extending incompressible encryption to incorporate anonymity. We demonstrated that existing incompressible SKE schemes can handle single encryption queries with inherent anonymity and enhanced these schemes using programmable PRFs to support unbounded encryption queries. This marks a significant advancement as the first non-trivial incompressible SKE capable of unlimited encryption queries. We modified Branco et al.'s rate-1 incompressible SKE scheme to achieve both anonymity and CPA security. Additionally, we showed that their hybrid encryption scheme could achieve anonymity using an anonymous programmable HPS.

Our work opens new research avenues, particularly investigating whether programmable PRFs are necessary for constructing incompressible SKE schemes capable of handling unbounded queries or if one-way functions could suffice. Another exciting direction is developing rate-1 anonymous incompressible SKE schemes in the random oracle model, balancing security and efficiency for practical applications.

This thesis significantly advances incompressible encryption, addressing key limitations and providing robust solutions for future cryptographic systems. We hope these findings inspire further research and development, contributing to a more secure and private digital landscape.

Bibliography

[ABN10] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In *Theory of Cryptography: 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings 7*, pages 480–497. Springer, 2010. 3

[BBDP01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 566–582. Springer, 2001. 3

[BDD22] Pedro Branco, Nico Döttling, and Jesko Dujmović. Rate-1 incompressible encryption from standard assumptions. In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part II*, pages 33–69. Springer, 2022. 4, 5, 8, 14, 32

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*, pages 523–540. Springer, 2004. 10

[Dzi06] Stefan Dziembowski. On forward-secure storage. In *Annual International Cryptology Conference*, pages 251–270. Springer, 2006. 4, 5, 7, 17

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016. 5

[GKRV24] Rishab Goyal, Venkata Koppula, Mahesh Sreekumar Rajasree, and Aman Verma. Incompressible functional encryption. Cryptology ePrint Archive, Paper 2024/798, 2024. https://eprint.iacr.org/2024/798. 5

[GM82] Shaft Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *14th Annual ACM Symposium on Theory of Computing, STOC 1982*, pages 365–377, 1982. 1

[GMP22] Paul Grubbs, Varun Maram, and Kenneth G Paterson. Anonymous, robust post-quantum public key encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 402–432. Springer, 2022. 3

[GWZ22] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Incompressible cryptography. In *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part I*, pages 700–730. Springer, 2022. 4, 5

[GWZ23] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Somewhere Randomness Extraction and Security against Bounded-Storage Mass Surveillance, 2023. Report Number: 409. 4, 5

[HLH+22] Zhengan Huang, Junzuo Lai, Shuai Han, Lin Lyu, and Jian Weng. Anonymous public key encryption under corruptions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 423–453. Springer, 2022. 3

[HSHI02] Goichiro Hanaoka, Junji Shikata, Yumiko Hanaoka, and Hideki Imai. Unconditionally secure anonymous encryption and group authentication. In *Advances in Cryptology—ASIACRYPT 2002: 8th International Conference on the Theory and Application of Cryptology and Information Security Queenstown, New Zealand, December 1–5, 2002 Proceedings 8*, pages 81–99. Springer, 2002. 3

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021. 5

[KKRS24] Venkata Koppula, Abhinav Kumar, Mahesh Sreekumar Rajasree, and Harihar Swaminathan. Incompressible encryption beyond cpa/cca security. *Manuscript submitted*, 2024. 5

[Moh10] Payman Mohassel. A closer look at anonymity and robustness in encryption schemes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 501–518. Springer, 2010. 3

[MW20] Tal Moran and Daniel Wichs. Incompressible encodings. In *Annual International Cryptology Conference*, pages 494–523. Springer, 2020. 5, 8, 12

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, 1990. 1

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming prfs, the lwe way. In *IACR International Workshop on Public Key Cryptography*, pages 675–701. Springer, 2018. 11

[RS91] Charles Rackoff and Daniel R Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Annual international cryptology conference*, pages 433–444. Springer, 1991. 1