Rate-1 Incompressible Encryption from Standard Assumptions

Pedro Branco[1], Nico Döttling[2], and Jesko Dujmović[2,3]

[1]Johns Hopkins University
[2]Helmholtz Center for Information Security (CISPA)
[3]Saarland University

Abstract

Incompressible encryption, recently proposed by Guan, Wichs and Zhandry (EUROCRYPT'22), is a novel encryption paradigm geared towards providing strong long-term security guarantees against adversaries with *bounded long-term memory*. Given that the adversary forgets just a small fraction of a ciphertext, this notion provides strong security for the message encrypted therein, even if, at some point in the future, the entire secret key is exposed. This comes at the price of having potentially very large ciphertexts. Thus, an important efficiency measure for incompressible encryption is the message-to-ciphertext ratio (also called the rate). Guan et al. provided a low-rate instantiation of this notion from standard assumptions and a rate-1 instantiation from indistinguishability obfuscation (iO). In this work, we propose a simple framework to build rate-1 incompressible encryption from standard assumptions. Our construction can be realized from, e.g. the DDH and additionally the DCR or the LWE assumptions.

1 Introduction

Incompressible Cryptography Incompressible cryptography [Dzi06b, DGO19, GLW20, MW20, GZ21, GWZ22] is a flourishing paradigm trying to leverage memory limitations of adversaries to achieve strong security goals. While traditionally, the goal of cryptography in the bounded storage model [Mau93] is to minimize the need for computational assumptions or even obtain information-theoretically secure constructions, incompressible cryptography is geared more toward mitigating the consequences of key exfiltration and key exposure attacks. In this work, we focus on the notion of *incompressible encryption* [Dzi06b, GWZ22] [1] recently coined by Guan et al. [GWZ22]. An incompressible encryption scheme produces large, incompressible ciphertexts and guarantees that any adversary who *forgets* even a small fraction of the ciphertext data will learn nothing about the encrypted data, even if he is later given the corresponding secret key!

One motivation for incompressible encryption is to hamper adversaries conducting a mass-surveillance operation by forcing them to store massive amounts of ciphertext data even if they are just interested in a tiny fraction of the encrypted data. In a similar scenario, an adversary trying to exfiltrate information encrypted under an incompressible encryption scheme from a data-center will have to exfiltrate massive amounts of data, even if his exfiltration target is just a small piece of information.

Notions orthogonal to incompressible encryption are encryption in the *bounded-retrieval model* [Dzi06a, DLW06, ADW09, ADN+10, HLWW13, BKR16, BD17, MW20] where the goal is to make the secret key large and incompressible (to make it hard to exfiltrate) while keeping all other system parameters small, such as the sizes of public keys and ciphertexts, as well as the overhead of encryption and decryption.

Encryption with High Rate. An important efficiency measure of encryption schemes is their ciphertext expansion or rate. The rate of an encryption scheme is the ratio between plaintext size and ciphertext size. The closer the rate is to 1, the more efficient a scheme manages to pack information into a ciphertext.

[1]Dziembowski [Dzi06b] introduced this concept under the name *forward-secure storage* in the symmetric key setting.

Conversely, the closer the rate is to 0, the less information is encoded in potentially large ciphertexts. For incompressible encryption, achieving a high rate (ideally converging to 1), especially if we think of the data center application above, where a small rate would also put a massive burden on the data center.

Guan et al. [GWZ22] provided two constructions of incompressible encryption.

- A construction from the minimal assumption of public-key encryption which has ciphertext-rate approaching 0.
- A construction from indistinguishability obfuscation (iO) [BGI+01, GGH+13, JLS21] which achieves ciphertext-rate approaching 1.

We remark that their rate-1 construction relies on non-black-box techniques and iO, which gives this result a strong feasibility flavor.

Given this state of affairs, this work is motivated by the following question:

Can we build a rate-1 incompressible encryption scheme based on standard assumptions while only making black-box use of cryptographic primitives?

1.1 Our Results

In this work, we build a rate-1 incompressible encryption scheme from standard assumptions while only using black-box techniques. Our result uses what we call programmable hash proof systems (HPS) (which are a variant of standard HPS [CS98, CS02] with some additional properties), plain-model incompressible encodings [MW20] and a pseudorandom generator (PRG). In particular, we prove the following theorem.

Theorem 1 (Informal)**.** *Let S be the storage capacity of the adversary and let n be the size of the encrypted messages. Assuming programmable HPS, incompressible encodings and PRGs exist, there is an incompressible encryption scheme fulfilling the following properties:*

1. *Ciphertexts are of size $n + n^{\varepsilon} \cdot \mathsf{poly}(\lambda)$ for some $\varepsilon > 0$.*
2. *The public key is of size $n^{\varepsilon'} \cdot \mathsf{poly}(\lambda)$ for some $\varepsilon' > 1/2$.*
3. *Moreover, the size of ciphertexts is only slightly larger than the adversary's storage space, that is, $S + \mathsf{poly}(\lambda)$.*

The ciphertext rate $n/(n + n^{\varepsilon} \cdot \mathsf{poly}(\lambda))$ approaches 1 for large enough messages. Additionally, the public key is sublinear in the size of the encrypted message.

In terms of assumptions, incompressible encodings can be based on either decisional composite residuosity (DCR) or learning with errors (LWE). The PRG can be based on any one-way function. We also show that programmable HPS can be instantiated from the decisional Diffie-Hellman (DDH) assumption by tweaking the famous HPS by Cramer and Shoup [CS02]. Consequently, our final incompressible encryption scheme can be based solely on standard assumptions.

Post-quantum construction. In our main construction we crucially use the DDH assumption. It is well-known that DDH can be broken by quantum adversaries, so our construction is prone to quantum attacks [Sho94].

To overcome this issue, we show that the HPS construction of [ADMP20], which is based on isogeny-based assumptions, is also a programmable HPS. The drawback of this construction is that it has large public keys. That is, the public key size grows linearly with the size of the message.

Recall that isogeny-based assumptions are presumably post-quantum secure. By plugging this HPS into our main construction, we obtain a classically secure incompressible encryption scheme only based on post-quantum assumptions.

This does not necessarily mean that our construction is secure against quantum adversaries as we only allow the adversary to compress into classical (non-quantum) memory. While we think limiting the adversary

to classical long-term storage is reasonable as long-term quantum storage seems to be even harder to achieve than quantum computation the scheme could still be insecure against quantum adversaries as demonstrated the cocurrent work of [LMQW22].

Streaming encryption. Streaming encryption/decryption is a property of incompressible encryption schemes which allows the honest encryptor/decryptor to perform operations with very low storage capacity. It is easy to see that streaming decryption is an inherently conflicting property with high rate ciphertexts [GWZ22]. This is because the honest decryptor needs storage at least as large as the size of the message. Otherwise, an adversary can essentially mimic the decryptor and learn something about the encrypted message (e.g., the most significant bit).

However, we note that our scheme has *stream encryption*, i.e., the honest encryptor does not need much space to perform encryption. This follows from the fact that the incompressible encodings construction of [MW20] has stream encoding.

Extension to CCA security. In the security experiment for incompressible encryption presented in [GWZ22] the adversary is never allowed to query a decryption oracle. In other words, their work only considered IND-CPA incompressible encryption. In this work, we also give the adversary access to an decryption oracle extending incompressible encryption to IND-CCA2 incompressible encryption. We stress that IND-CCA2 security is usually considered the *right* security definition to use in practice. We show that our construction is, in fact, is IND-CCA2 incompressible secure.

Focus on the Plain Model. We demonstrate a concretely and asymptotically more efficient construction in the random oracle model (ROM). However, we also show that incompressible encryption, which is secure in the ROM, might be prone to attacks as soon as we instantiate the random oracle by a specific hash function. Similarly to [CGH04, Den02, GK03, BBP04, MRH04, Bla06, BFM15, GKW17], we construct scheme that can be proven secure in the ROM. However, when we instantiate the random oracle, the scheme turns out to be insecure. These observations support our focus on the plain model.

1.2 Comparison with Previous Work

As mentioned previously, incompressible encryption was introduced in [GWZ22] where two schemes are presented. The first one is based only on the minimal assumption of PKE. However, the ciphertext rate is very far from 1. The second one achieves rate-1 but is based on $\mathrm{i}\mathcal{O}$. We compare these schemes in Table 1.

	Ciphertext Rate	Public key Size	Hardness Assumption	Security
[GWZ22]	$1/\mathsf{poly}(\lambda)$	$\mathsf{poly}(\lambda)$	PKE	IND-CPA
[GWZ22]	1	$\mathsf{poly}(\lambda)$	$\mathrm{i}\mathcal{O}$	IND-CPA
Our result	1	$n^{\varepsilon'} \cdot \mathsf{poly}(\lambda)$	DDH + {DCR,LWE}	IND-CCA2
Our result	1	$n \cdot \mathsf{poly}(\lambda)$	isogeny-based + {DCR,LWE}	IND-CCA2
Our result	1	$n^{\epsilon'} \cdot \omega(\mathsf{polylog}(\lambda))$	LWE	IND-CPA

Table 1: Comparison with previous work. Here, n denotes the size of the encrypted messages and ε' is any constant between 1/2 and 1.

Other related work. Some recent works made significant progress in the area of incompressible cryptography. The works of [DGO19, GLW20, MW20] proposed constructions for incompressible encodings either

in the random oracle model or in the CRS model. The work of [GZ21] used the BSM together with computational assumptions to propose constructions of primitives that are not known just from computational assumptions, such as virtual grey-box obfuscation.

Incompressible cryptography is closely related to the bounded storage model (BSM) [Mau92]. However, most works in the BSM (e.g. [CM97, AR99, Raz17, GZ19, DQW21]) focus on achieving unconditional security for primitives that are already known from computational assumptions such as public-key encryption and oblivious transfer.

Open Problems. We leave the open problem of developing an incompressible encryption scheme that combines concretely short public keys with small ciphertexts. A possible approach for this would be to find a programmable hash proof system where the size of the public key is essentially independent of the size of the encapsulated key.

It might also be interesting if our construction only using post-quantum assumption is post-quantum secure. If this is not the case, a construction that is also post-quantum secure is of interest.

Acknowledgement. We would like to thank Stefan Dziembowski for discussions and comments. We would also like to thank Daniel Wichs for comments and pointing out the LWE HPS.

Pedro Branco: Part of this work was done while at IST University of Lisbon.

Funding statement for Nico Döttling: Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. (ERC-2021-STG 101041207 LACONIC)

2 Technical Overview

In this technical overview, we sketch the main techniques to build an IND-CPA incompressible scheme. We later argue how these techniques can be tweaked to obtain a scheme that is IND-CCA2 incompressible secure.

Security Notion The syntax and correctness notions for incompressible encryption are identical to standard public-key encryption (PKE). The main difference is in the security definition. Since the security notion of incompressible encryption is relatively new, we will briefly detail its security experiment here. Consider the following security game between a challenger $\mathcal{C}$ and a 3-stage PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$.

1. $\mathcal{C}$ creates a pair of public and secret keys pk, sk.
2. Given pk, the first stage $\mathcal{A}_1$ chooses two messages $\mathsf{m}_0, \mathsf{m}_1$.
3. $\mathcal{C}$ chooses $b \leftarrow\$ \{0,1\}$ uniformly at random and encrypts $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_b)$.
4. Given the ciphertext ct and the state of $\mathcal{A}_1$, the second stage $\mathcal{A}_2$ produces a state st of size $S < |\mathsf{ct}|$. That is, the state st should be somewhat smaller than ct.
5. Now, the third stage $\mathcal{A}_3$ receives as input the state st (produced by $\mathcal{A}_2$) and the secret key sk. The goal of $\mathcal{A}_3$ is to guess the bit b.

We say that an incompressible encryption is secure if, for any adversary, $\mathcal{A}$ the advantage of winning the following game is negligible in the security parameter λ.

2.1 The Scheme of GWZ

Before we provide an outline of our construction, we will briefly discuss the underlying ideas of the low-rate incompressible encryption scheme constructed in [GWZ22]. At the very core is the following idea: The ciphertext essentially consists of a very long truly random random string R and a short payload part $c = (c_1, c_2)$, where c_1 is an encryption of a seed k for a randomness extractor Ext, and $c_2 = \mathsf{Ext}(k, R) \oplus \mathsf{m}$ is essentially a one-time-pad encryption of the message m under the key $\mathsf{Ext}(k, R)$. Clearly, if c_1 was *not* part of the ciphertext, then security of this scheme follows routinely by the following observations:

- In the view of the third stage $\mathcal{A}_3$ of the adversary R has high min-entropy, as R is uniformly random and the state st is significantly shorter than R.
- Furthermore, as we assume c_1 is not part of the ciphertext, st is independent of k
- Hence by the extraction property of Ext the string $\mathsf{Ext}(k, R)$ is uniformly random in the adversary's view, and therefore m_b is statistically hidden.

Now, the main idea of [GWZ22] to make this approach work even though c_1 is part of the ciphertext is to encrypt k in such a way that c_1 *can be made independent* of the extractor seed k. This is achieved by choosing a suitable encryption scheme for which c_1 can be chosen independently of k, and a suitable secret key which decrypts c_1 to k can be chosen *after the fact*, i.e. after the leakage st has been computed. [GWZ22] provide an elegant construction of such a scheme from non-compact single-key functional encrytion, which can be built from any public key encryption scheme [GVW12].

2.2 The Big Picture

While our construction departs significantly from the blueprint of [GWZ22] we use the same high-level concept of an encryption scheme that allows delaying secret-key generation in the security proof. Rather than constructing incompressible PKE directly, we first tackle the intermediate and simpler task of realizing a rate-1 *incompressible symmetric-key encryption*. In a second step, we will then transform *any* incompressible SKE scheme into an incompressible PKE scheme in a rate-preserving way. It turns out that even constructing a rate-1 incompressible SKE from standard assumptions is a non-trivial task and does not follow, e.g. from the (low-rate) public-key construction of [GWZ22].

Since our two steps are independent of one another, improvements of either in future work will lead to better incompressible encryption schemes. For simplicity, in the following outline, we will focus only on CPA security, whereas in the main body, we present a CCA secure construction.

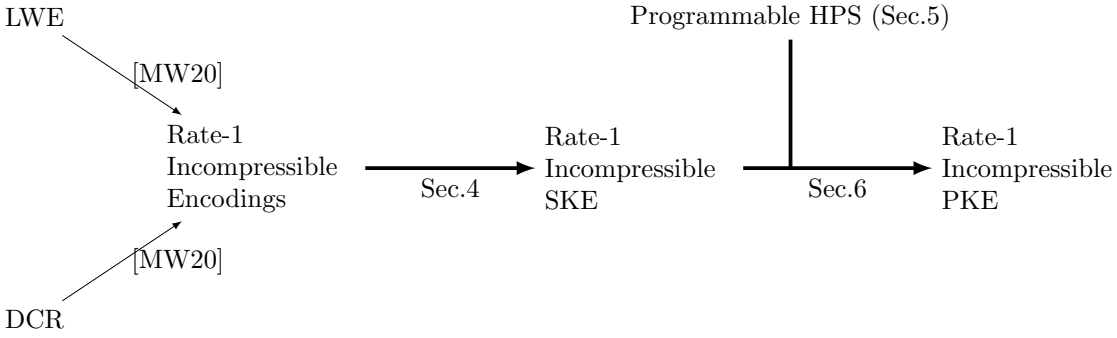

Figure 1: Overview of the results in this work, bold arrows are contributions of this work.

2.3 Rate-1 Incompressible Symmetric-Key Encryption

In the symmetric-key setting, the syntax and correctness of incompressible SKE are pretty much that of standard symmetric-key encryption, whereas the security notion is similar to that of incompressible PKE,

just with the difference that the first stage $\mathcal{A}_1$ of the adversary is not given a public key (as there is none). Thus, the security notion we consider here is the incompressible encryption-analogue of security against an eavesdropper (IND-EAV).

Failed Naive Attempts. As a (failed) very first attempt, one may try "make work" an incompressible SKE construction from the One-Time-Pad (OTP), i.e. the secret key k is a random bit-string as long as the message m and the ciphertext is $c = \mathsf{k} \oplus \mathsf{m}$. However, the obvious issue with this is that such a ciphertext c *decomposes* into many one-bit ciphertexts $c_i = \mathsf{k}_i \oplus \mathsf{m}_i$, and it is enough for $\mathcal{A}_2$ to leak a few bits of c to enable $\mathcal{A}_3$ to partially decrypt c and thus distinguish encryptions of m_0 from encryptions of m_1. As a next idea, one may try the following: Encryption chooses a (fresh) pseudorandom generator (PRG) seed s, encrypt m into $\hat{\mathsf{m}} = \mathsf{m} \oplus \mathsf{PRG}(\mathsf{s})$, use k to encrypt the seed s into a *header ciphertext* c, i.e. the encryption of m is $(\mathsf{Enc}(\mathsf{k}, \mathsf{s}), \mathsf{m} \oplus \mathsf{PRG}(\mathsf{s}))$. While this approach does look promising, we observe that it is stuck at either *leakage-rate* $1/2$ or ciphertext-rate $1/2$, that is as soon as $\mathcal{A}_3$ learns $\mathsf{Enc}(\mathsf{k}, \mathsf{s})$ in its entirety and a few bits of $\mathsf{m}_b \oplus \mathsf{PRG}(\mathsf{s})$, he will be able to distinguish encryptions of m_0 from m_1.

Introducing Circularity. Clearly, we need some kind of mechanism to *glue* the two ciphertexts components together, i.e. we want to make it such that if some parts of $\hat{\mathsf{m}}$ are missing, then c will be useless (and vice versa). As a first, heuristic "hands-on" approach to achieve this, we can try to use $\hat{\mathsf{m}}$ as a source of randomness from which we extract a key to mask the seed s. Thus, let $\mathsf{Ext}(\cdot, \cdot)$ be a *seeded* randomness extractor. We compute a ciphertext $(c, \hat{\mathsf{m}})$ by first computing $\hat{\mathsf{m}} = \mathsf{m} \oplus \mathsf{PRG}(\mathsf{s})$ for a random seed s as before, but then encrypt s into c via $c = \mathsf{s} \oplus \mathsf{Ext}(\mathsf{k}, \hat{\mathsf{m}})$, i.e. we use k as an extractor seed to extract a one-time-pad key $\mathsf{Ext}(\mathsf{k}, \hat{\mathsf{m}})$ from $\hat{\mathsf{m}}$. Clearly, given k and a ciphertext $(c, \hat{\mathsf{m}})$, we can decrypt by first computing $s = c \oplus \mathsf{Ext}(\mathsf{k}, \hat{\mathsf{m}})$ and then $\mathsf{m} = \hat{\mathsf{m}} \oplus \mathsf{PRG}(\mathsf{s})$. The rationale for why we hope this construction to be secure is that as soon as a significant fraction of the bits of $\hat{\mathsf{m}}$ are lost, the output of the extractor $\mathsf{Ext}(\mathsf{k}, \hat{\mathsf{m}})$ should look uniform, and thus $\hat{\mathsf{m}} = \mathsf{m} \oplus \mathsf{PRG}(\mathsf{s})$ should hide m by the pseudorandomness of PRG. However, this circularity backfires when trying to establish security of this construction just from the pseudorandomness of PRG and the randomness-extraction property of Ext: In order to use pseudorandomness of PRG, we first need to *remove* the s from the view of the adversary, but $c = \mathsf{s} \oplus \mathsf{Ext}(\mathsf{k}, \hat{\mathsf{m}})$ is correlated with s given k. On the other hand, in order to use the randomness extraction property of Ext we need that $\hat{\mathsf{m}}$ has high entropy given st. But all the entropy of $\hat{\mathsf{m}} = \mathsf{m} \oplus \mathsf{PRG}(\mathsf{s})$ comes from the seed s, which is very small. Hence $\approx \lambda$ bits of $\hat{\mathsf{m}}$ suffice to information-theoretically determine s.

Consequently, while heuristically, this construction seems secure, it seems unlikely the individual security properties of PRG and Ext suffice to *prove* this construction secure.

Breaking Circularity Hence, what we need is a mechanism to break the circularity, which we have just introduced. Looking at where establishing security of the above construction gets stuck, a natural point to start is to make it such that $\hat{\mathsf{m}}$ looks like it has a large amount of real entropy once a few bits of $\hat{\mathsf{m}}$ are missing, i.e. $L(\hat{\mathsf{m}})$ being computationally indistinguishable from $L(\hat{r})$ for a high-entropy distribution $\hat{r}$ for any efficiently computable leakage function $L(\cdot)$ [2].

Incompressible encodings. Fortunately, an encoding mechanism achieving this notion called *incompressible encodings* was just recently introduced and constructed by Moran and Wichs [MW20]. As a technical tool, they introduced the notion of *HILL-entropic encoding* in their work, which will be sufficient, if not to say ideally suited for our construction. Such a scheme consists of an encoding algorithm En and a decoding algorithm De, both of which rely on a (large) common random string $\mathsf{crs} \leftarrow\!\!\$\, \{0,1\}^t$:

- The encoding algorithm $\mathsf{En}_{\mathsf{crs}}(m)$ is a *randomized algorithm* which takes a message m and produces an encoding $\hat{m}$
- The decoding algorithm $\mathsf{De}_{\mathsf{crs}}(\hat{m})$ is a deterministic algorithm which takes an encoding $\hat{m}$ and returns a message m.

[2] In our case the leakage function L is described by the adversary's second stage $\mathcal{A}_2$

In terms of correctness, one naturally requires that encoding followed by decoding leads to the original message. Security-wise, we require that there exists a simulator Sim which on input a message m produces a pair $(\mathsf{crs}', \tilde{m})$, which is computationally indistinguishable from a real pair of crs and encoding of m, i.e.

$$(\mathsf{crs}, \mathsf{En}_{\mathsf{crs}}(m)) \approx_c \mathsf{Sim}(m),$$

where $\mathsf{crs} \leftarrow\!\!\$\, \{0,1\}^t$. Additionally, we require that if $(\mathsf{crs}', \tilde{m}) \leftarrow \mathsf{Sim}(m)$, then $\tilde{m}$ has almost full *true* min-entropy given crs', i.e. $\tilde{H}_\infty(\tilde{m}|\mathsf{crs}') \geq (1-\epsilon)n$, where $\tilde{H}_\infty$ is the average conditional min-entropy. The (very) high level idea how this can be achieved is that in simulation the common random string and the encoded message *switch roles*, in the sense that the simulated common random string crs' encodes the message m, whereas the encoding $\tilde{m}$ now has room to have (very) high entropy.

Moran and Wichs [MW20] provide two instantiations of their construction, one from DCR and one from LWE. These constructions achieve rate-1, i.e., the encoding is only slightly larger than the encoded message. The conceptual idea behind the construction is rather elegant: The encoding function $\mathsf{En}_{\mathsf{crs}}(m)$ generates a pair of public key and trapdoor $(\mathsf{pk}, \mathsf{td})$ of *preimage-sampleable surjective lossy function* F (for which we have efficient constructions from DCR or LWE) and sets x to be a randomly sampled preimage of $m \oplus \mathsf{crs}$, i.e. $x = F_{\mathsf{td}}^{-1}(m \oplus \mathsf{crs})$, and sets $\hat{m} = (\mathsf{pk}, x)$. To decode $\hat{m}$, one computes $m = F_{\mathsf{pk}}(x) \oplus \mathsf{crs}$. The simulator Sim chooses a *highly lossy public key* $\tilde{\mathsf{pk}}$, chooses x uniformly at random, and sets $\mathsf{crs}' = m \oplus F_{\tilde{\mathsf{pk}}}(x)$ and $\tilde{m} = (\tilde{\mathsf{pk}}, x)$. Given that F_{pk} is *regular* for surjective keys pk, meaning that if x is uniform then $F_{\mathsf{pk}}(x)$ is also (statisticalluy close to) uniform, we can routinely establish that real pairs $(\mathsf{crs}, \hat{m})$ are computationally indistinguishable from simulated $(\mathsf{crs}', \tilde{m})$ using the indistinguishability of surjective public keys pk and highly lossy public keys $\tilde{\mathsf{pk}}$. Moreover, for simulated pairs $(\mathsf{crs}' = m \oplus F_{\tilde{\mathsf{pk}}}(x), \tilde{m} = (\tilde{\mathsf{pk}}, x))$ we can easily argue that x (and hence $\tilde{m}$) has high min-entropy given $\mathsf{crs}' = m \oplus F_{\tilde{\mathsf{pk}}}(x)$, as $F_{\tilde{\mathsf{pk}}}$ is highly lossy and hence x has high min entropy given $F_{\tilde{\mathsf{pk}}}(x)$.

Moran and Wichs [MW20] go on to show that for any incompressible encoding/HILL-entropic encoding, the common random string crs must be as long as the message, if one wants to establish security from a *falsifiable assumption* [Nao03] under a black-box reduction.

The Full Construction. We will now provide our complete construction of incompressible SKE and sketch the security proof. For our scheme, the secret key K is a uniformly random bit-string of suitable length which will be parsed as $\mathsf{K} = (\mathsf{crs}, \mathsf{k})$, where crs is the common random string for a HILL-entropic encoding $(\mathsf{En}, \mathsf{De})$, and k is the seed for a randomness extractor Ext. Encryption and decryption work as follows.

- $\mathsf{Enc}(\mathsf{K} = (\mathsf{crs}, \mathsf{k}), \mathsf{m})$: Choose a uniformly random PRG seed $\mathsf{s} \leftarrow\!\!\$\, \{0,1\}^\lambda$ and compute $\hat{\mathsf{m}} = \mathsf{En}_{\mathsf{crs}}(m \oplus \mathsf{PRG}(\mathsf{s}))$. Compute $c = \mathsf{s} \oplus \mathsf{Ext}(\mathsf{k}, \hat{\mathsf{m}})$ and output the ciphertext $\mathsf{ct} = (c, \hat{\mathsf{m}})$.
- $\mathsf{Dec}(\mathsf{K} = (\mathsf{crs}, \mathsf{k}), \mathsf{ct} = (c, \hat{\mathsf{m}}))$: Compute $\mathsf{s} = c \oplus \mathsf{Ext}(\mathsf{k}, \hat{\mathsf{m}})$ and output $m = \mathsf{De}_{\mathsf{crs}}(\hat{\mathsf{m}}) \oplus \mathsf{PRG}(\mathsf{s})$.

Correctness of this scheme follows routinely.

Security of this scheme is established along the following lines. First we rely on the security of the HILL-entropic encoding to replace $(\mathsf{crs}, \hat{\mathsf{m}})$ with a *simulated pair* $(\mathsf{crs}', \tilde{\mathsf{m}}) = \mathsf{Sim}(\mathsf{m} \oplus \mathsf{PRG}(\mathsf{s}))$. By the security of the HILL-entropic encoding, this modification is (computationally) unnoticeable to the adversary. However, now the encoding $\tilde{\mathsf{m}}$ has true high min-entropy given crs'. Thus, using a min-entropy chain rule (e.g. by [DORS08]) we can argue that $\tilde{\mathsf{m}}$ still has sufficiently high min-entropy given both crs' and a leak $L(\tilde{\mathsf{m}})$. Hence, the randomness extraction property guarantees that $\mathsf{Ext}(\mathsf{k}, \tilde{\mathsf{m}})$ will extract uniform randomness (given crs' and $L(\tilde{\mathsf{m}})$). To establish this we need a mild extra property of the extractor Ext that given a (uniformly random) extractor output y and $\tilde{\mathsf{m}}$ we can sample a key k' *after the fact* such that $(\mathsf{k}', y) \approx (\mathsf{k}, \mathsf{Ext}(\mathsf{k}, \tilde{\mathsf{m}}))$. Hence in the next hybrid modification, we can thus replace $c = \mathsf{s} \oplus \mathsf{Ext}(\mathsf{k}, \tilde{\mathsf{m}})$ with a uniformly random and independent string c'. Now that c' is independent of s, we can use the pseudorandomness property of PRG to replace $\mathsf{m} \oplus \mathsf{PRG}(\mathsf{s})$ in $(\mathsf{crs}', \tilde{\mathsf{m}}) = \mathsf{Sim}(\mathsf{m} \oplus \mathsf{PRG}(\mathsf{s}))$ with a uniformly random string u, i.e. $(\mathsf{crs}', \tilde{\mathsf{m}}) = \mathsf{Sim}(u)$. We have finally arrived at an experiment where the ciphertext $\mathsf{ct} = (c', \tilde{\mathsf{m}})$ is independent of the message m, and hence the adversary's advantage is 0.

Concerning the rate of this scheme, note that a ciphertext $\mathsf{ct} = (c, \hat{\mathsf{m}})$ has rate 1, as c is just of size $\mathsf{poly}(\lambda)$ (independent of the message length n), and the HILL-entropic encoding $\hat{\mathsf{m}}$ is rate 1.

2.4 From Symmetric-Key to Public-Key Incompressible Encryption via Hash Proof Systems

Now that we have a construction of incompressible SKE, we need a way to establish a *long* key K between the sender and receiver. This is a job for a *key encapsulation mechanism* (KEM) [CS03]. A key-encapsulation mechanism consists of:

- A key-encapsulation mechanism consists of a key-generation algorithm KeyGen which produces a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$.
- An encapsulation algorithm which takes a public key pk and produces a symmetric key K and a ciphertext header c_0 *encapsulating* K.
- A decapsulation algorithm Dec which takes a secret key sk and a ciphertext header c_0 and outputs a key K.

The correctness requirement is the obvious one, whereas the standard security requirement is that K is pseudorandom given pk and c_0. A symmetric key K generated via a KEM can now be used to encrypt a message m into a payload ciphertext c_1 using a symmetric key encryption scheme. The full ciphertext is $c = (c_0, c_1)$.

However, to transform an incompressible SKE into an incompressible PKE *not just any key encapsulation mechanism will do*. The simple reason is that in the incompressible (public key) encryption security game, the adversary gets to see the secret key sk in the end, which will allow him to decapsulate the (short) ciphertext header c_0 into the symmetric key K. But the standard security notion of KEMs discussed above does not require that the encapsulated key K follows a uniform distribution. Indeed, e.g. for simple PRG-based KEMs, the encapsulated key is statistically far from uniform. However, recall that in our construction of incompressible SKE above, we made critical use of the fact that the key K follows a uniform distribution and that the security reduction can *program* it in a suitable way.

Thus, we need a KEM which we can switch into a mode in which the ciphertext header c_0 encapsulates a truly uniform key K. As we need the ciphertext header c_0 to be substantially shorter than the encapsulated key K, the entropy of K in this mode *must* come from the secret key sk.

Enter Hash proof systems. This is where hash proof systems (HPS) [CS02] come into play [3]. Recall that HPS are defined relative to an NP-language $\mathcal{L} \subseteq \{0,1\}^k$. We have a key-generation algorithm KeyGen which generates a public or *projected* key pk, and a secret or *hashing* key sk. The hashing or decapsulation algorithm Decap takes the secret key sk and *any* $x \in \{0,1\}^k$ and produces a hash value K. The restricted hashing or encapsulation algorithm Encap takes a public key pk, an $x \in \mathcal{L}$ and a witness w (with respect to a fixed NP-relation for $\mathcal{L}$) for membership of x in $\mathcal{L}$ and produces a hash-value K.

In terms of correctness or completeness, we require that Decap and Encap agree on $\mathcal{L}$, i.e. if $x \in \mathcal{L}$ and w is a valid witness for x, then it holds that $\mathsf{Decap}(\mathsf{sk}, x) = \mathsf{Encap}(\mathsf{pk}, x, w)$.

In terms of security, we require *smoothness*, namely given that $x \notin \mathcal{L}$, it holds that $\mathsf{Decap}(\mathsf{sk}, x)$ is statistically close to uniform *given* pk.

HPS are especially useful for sparse pseudorandom languages $\mathcal{L}$, such as the decisional Diffie-Hellman (DDH) language) [CS02]. We define this language with respect to a pair of (randomly chosen) generators $g, h \in \mathbb{G}$, where $\mathbb{G}$ is a cryptographic group of prime order p. A pair $x = (g', h')$ is in $\mathcal{L}$, if there exists an $r \in \mathbb{Z}_p$ such that $g' = g^r$ and $h' = h^r$. The DDH assumption states that a random element in $\mathcal{L}$, i.e. a pair (g^r, h^r) is computationally indistinguishable from a pair of uniformly random group elements (u, v) [4]

[3] HPS have been instrumental in many prior works on leakage resilience cryptography e.g. [ADN+10, HLWW13]

[4] Note that such a pair is not in $\mathcal{L}$, except with negligible probability $1/p$.

In the Cramer-Shoup [CS02] scheme, the secret key $\mathsf{sk} = (\alpha, \beta)$ consists of two uniformly random values $\alpha, \beta \in \mathbb{Z}_p$, and the public key pk is computed as $\mathsf{pk} = g^\alpha h^\beta$. Given a public key pk an instance $c_0 = (g^r, h^r)$ with witness r, we compute a key $\mathsf{K} = \mathsf{pk}^r$. Given a secret key $\mathsf{sk} = (\alpha, \beta)$ and an instance $c_0 = (g', h')$ we compute a key $\mathsf{K} = g'^\alpha h'^\beta$. It follows routinely that encapsulation and decapsulation agree on $\mathcal{L}$. Moreover, for a $(g^*, h^*) \notin \mathcal{L}$ it holds $\mathsf{K}^* = g^{*\alpha} h^{*\beta}$ is uniformly random given $\mathsf{pk} = g^\alpha h^\alpha$ by a simple linear algebra argument.

Hash Proof Systems, and in particular the Cramer-Shoup HPS (almost) give us a KEM with the desired properties. Namely, given pk and (g, h), to encapsulate a key k we choose a uniformly random $r \in \mathbb{Z}_p$ and compute $c_0 = (g^r, h^r)$ and $\mathsf{K} = \mathsf{pk}^r$. To decapsulate K from $c_0 = (g', h')$ given $\mathsf{sk} = (\alpha, \beta)$, we compute $\mathsf{K} = g'^\alpha h'^\beta$.

A typical proof-strategy using HPS lets a reduction compute the encapsulated key (on the sender's side) via the decapsulation algorithm using the secret key. Correctness of the HPS ensures that this does not change K. Hence this modification will not be detected by an adversary. Now we don't need the witness r anymore. We can replace c_0 with a uniformly random c_0' and argue that this modification is computationally undetectable by the adversary, thanks to the DDH assumption. Since now c_0' is outside of $\mathcal{L}$ w.o.p, it holds that K is uniform even given pk, as desired.

However, this is still not enough to make our security reduction go through. It turns out we not only have to ensure that K is uniform given pk, but also that for *any given* K and fixed pk and c_0 we can find a secret key sk (compatible with pk) such that $\mathsf{Decap}(\mathsf{sk}, c_0) = \mathsf{K}$. Realizing this property using the Cramer-Shoup HPS directly seems hard, as in order to sample an $\mathsf{sk} = (\alpha, \beta)$ with $\mathsf{K} = g'^\alpha h'^\beta$ we would need to compute a discrete logarithm of K.

Programmable Hash Proof Systems For this purpose, we will consider a notion of *programmable* hash proof systems, which obey a stronger smoothness notion. In short, such an HPS has the following property. Given a public key pk, a (fake) ciphertext header c_0^* (not in $\mathcal{L}$) and secret auxiliary information aux depending on both pk and c_0, we can sample a uniformly random secret key sk^* such that $\mathsf{Decap}(\mathsf{sk}^*, c_0^*) = \mathsf{K}$, for which it holds that $(\mathsf{pk}, c_0^*, \mathsf{sk}^*) \approx_s (\mathsf{pk}, c_0^*, \mathsf{sk})$ if K is chosen uniformly random.

Our idea to achieve this is simple: We will concatenate Decap (and also Encap) with a *balanced small range* hash function $\mathsf{HC} : \mathbb{G} \to \{0,1\}$, i.e. we have $\mathsf{Decap}'(\mathsf{sk}, c_0) = \mathsf{HC}(\mathsf{Decap}(\mathsf{sk}, c_0))$ and $\mathsf{Encap}'(\mathsf{pk}, c_0, r) = \mathsf{HC}(\mathsf{Encap}(\mathsf{pk}, c_0, r))$. Here balanced means that if $h \in \mathbb{G}$ is a uniformly random group element, then $\mathsf{HC}(h)$ is statistically close to a uniformly random bit. While there exist deterministic constructions of such extractors for certain groups (e.g. [CFPZ09]) we can find such an HC for any group via the leftover-hash lemma [HILL99]. For such a hash function, we can efficiently sample a uniformly random pre-image $h \in \mathbb{G}$ of K for which we *do know* the discrete logarithm (with respect to a generator $g \in \mathbb{G}$). We achieve this via rejection sampling: Given a bit $\mathsf{K} \in \{0,1\}$, choose a uniformly random $z \in \mathbb{Z}_p$ and test whether $\mathsf{HC}(g^z) = \mathsf{K}$ (which happens with probability $1/2$), and reject and resample if the test fails.

Now let $h = g^y$, $\mathsf{pk} = g^t$ and $c_0^* = (g' = g^r, h' = g^s)$ be a public key and (fake) ciphertext, for which the auxiliary information is (y, t, r, s), i.e. the discrete logarithms of pk and c_0^*. Given a key $\mathsf{K} \in \{0,1\}$, we first sample a uniformly random $z \in \mathbb{Z}_p$ such that $\mathsf{HC}(g^z) = \mathsf{K}$. Now we have 2 linear constraints (over $\mathbb{Z}_p$) on $\mathsf{sk} = (\alpha, \beta) \in \mathbb{Z}_p^2$, namely

$$t = \alpha + \beta \cdot y$$

from $\mathsf{pk} = g^\alpha \cdot h^\beta$ and

$$z = \alpha r + \beta s$$

from $\mathsf{HC}(g^z) = \mathsf{HC}(g'^\alpha \cdot h'^\beta)$. Since we now have two equations and two unknowns α and β, we can solve for α and β using basic linear algebra.

We do pay a price to get programmability: Instead of getting $\log(|\mathbb{G}|)$ key bits per public key pk, we only get a single bit. Naturally, this can be improved up to $\log(\lambda)$ key-bits while keeping the above rejection sampling procedure expected polynomial time.

The Full Construction We are now ready to present our fully-fledged construction. This construction will have a large public key. We will later discuss how the size of the public key can be reduced.

Assume thus that $(\mathsf{Enc}, \mathsf{Dec})$ is an incompressible SKE scheme, and that $(\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ is a programmable HPS for a decision-membership-hard language $\mathcal{L}$, for concreteness assume the DDH language. Our incompressible PKE construction is given by the following algorithms.

- The key-generation algorithm KeyGen' generates random group elements $g, h \in \mathbb{G}$ and n pairs of public and secret keys $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n)$ using KeyGen (on g, h) and set $\mathsf{PK} = (g, h, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and $\mathsf{SK} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.
- The encryption algorithm Enc' proceeds as follows, given a public key $\mathsf{PK} = (g, h, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and a message m. First, generate a random DDH instance $c_0 = (g' = g^r, h' = h^r)$ using a random $r \leftarrow\!\!\$\, \mathbb{Z}_p$. Now compute the key-bits $\mathsf{K}_1 = \mathsf{Encap}(\mathsf{pk}_1, c_0, r), \ldots, \mathsf{K}_n = \mathsf{Encap}(\mathsf{pk}_n, c_0, r)$ and set $\mathsf{K} = (\mathsf{K}_1, \ldots, \mathsf{K}_n)$. Next, we use K to encrypt m using the incompressible SKE scheme, i.e. we compute $c_1 = \mathsf{Enc}(\mathsf{K}, \mathsf{m})$ and output the ciphertext $c = (c_0, c_1)$.
- The decryption algorithm Dec' takes a secret key $\mathsf{SK} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and a ciphertext $c = (c_0, c_1)$, and proceeds as follows. First, it decapsulates the key $\mathsf{K} = (\mathsf{K}_1, \ldots, \mathsf{K}_n)$ by computing $\mathsf{K}_1 = \mathsf{Decap}(\mathsf{sk}, c_0)$, $\ldots$, $\mathsf{K}_n = \mathsf{Decap}(\mathsf{sk}_n, c_0)$. Next, it decrypts c_1 to m via $\mathsf{m} = \mathsf{Dec}(\mathsf{K}, c_1)$.

Correctness of this scheme follows routinely from the correctness of its components.

Note that if the incompressible SKE scheme $(\mathsf{Enc}, \mathsf{Dec})$ is rate-1, then so is our public-key scheme $(\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$, as the only additional information in ciphertexts $c = (c_0, c_1)$ is the header c_0, which consists of just two group elements. On the other hand, note that the size of the public key of this scheme scales with the size n of the symmetric key K, which in our symmetric-key construction scales with the size of the message m.

Security of the Full Construction We will now turn to sketching the security proof for the main construction. In the first hybrid step (somewhat expectedly), we use the HPS Decap algorithm instead of the Encap algorithm to compute the key-bits K_i *in the encryption of the challenge ciphertext*. That is, in the encryption of the challenge ciphertext we replace $\mathsf{K}_i = \mathsf{Encap}(\mathsf{pk}_i, c_0, r)$ with $\mathsf{K}_i = \mathsf{Decap}(\mathsf{sk}_i, c_0)$ for all $i = 1, \ldots, n$. Due to the correctness property of the HPS, this modification does not change the distribution of K. Hence this hybrid change goes unnoticed by the adversary. In the second hybrid step, since we don't need r anymore, we replace $c_0 = (g^r, h^r)$ with a uniformly random c_0'. We can use the DDH assumption to argue that this modification goes unnoticed.

The next hybrid step is the critical one: We choose g, h, the pk_i and c_0' with auxiliary information, i.e. together with their discrete logarithms with respect to g, choose $\mathsf{K} \leftarrow\!\!\$\, \{0,1\}^n$ uniformly at random and sample each sk_i such that $\mathsf{K}_i = \mathsf{Decap}'(\mathsf{sk}_i, c_0')$ using the programming algorithm of the programmable HPS. We can argue statistical indistinguishability using the programmability property of HPS. The crucial observation now is that the public key $\mathsf{PK} = (g, h, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and the ciphertext header c_0 are computed *independently* of K and SK, and in fact we choose SK depending on K, i.e. we can choose SK after everything else.

This now allows us to turn an adversary $\mathcal{A}$ with non-negligible advantage in this hybrid experiment into an adversary $\mathcal{A}'$ with the same advantage against the incompressible SKE scheme. $\mathcal{A}'$ first generates PK as in the hybrid experiment and provides PK to the first stage $\mathcal{A}_1$ of $\mathcal{A}$, which will output m_0, m_1. Now the second stage $\mathcal{A}_2'$ gets to see a symmetric-key encryption c_1 of m_b, and turns this into a public-key encryption by setting $c = (c_0, c_1)$, where c_0 computed as in the hybrid experiment. This ciphertext c is then given $\mathcal{A}_2$, which outputs a state/leak st, and $\mathcal{A}_2'$ outputs the same state st.

Finally, $\mathcal{A}_3'$ given a symmetric key K and the state st proceeds as follows. Using the auxiliary information aux[5]. and the key K, it samples a secret key $\mathsf{SK} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ such that for all $i = 1, \ldots, n$ it holds that

[5]There is a technical subtlety in the security definition of incompressible SKE which we omitted before: We allow the first stage $\mathcal{A}_1'$ of a symmetric-key adversary $\mathcal{A}'$ to produce a large state (i.e. scaling with the message size), which is provided to

$\mathsf{K}_i = \mathsf{Decap}'(\mathsf{sk}_i, c_0)$, as in the hybrid experiment. Then, $\mathcal{A}'_3$ runs $\mathcal{A}_3$ on SK and st and outputs whatever $\mathcal{A}_3$ outputs.

It is not hard to see that from the view of $\mathcal{A}$, $\mathcal{A}'$ simulates the hybrid experiment perfectly. Hence, the advantage of $\mathcal{A}'$ against the incompressible symmetric-key security experiment is the same as that of $\mathcal{A}$ against the hybrid experiment, and we derive the desired contradiction.

Reducing the Public-Key-Size. As mentioned above, the construction we discussed in the last two paragraphs has a near-optimal ciphertext size (i.e. increasing the size of the symmetric-key ciphertext only by two group elements). In contrast, it has a very large public key which scales *linearly* with the size of the encrypted messages/the ciphertexts.

We will now discuss a tradeoff which achieves a better balance between ciphertext size and public key size. Concretely, we will provide a tradeoff which achieves a ciphertext size of $n + n^{\epsilon}\mathsf{poly}(\lambda)$ for an $0 < \epsilon < 1$ and public key size $n^{\epsilon'}\mathsf{poly}(\lambda)$ for an $1/2 < \epsilon' < 1$. I.e. we achieve ciphertext rate $1 - n^{\epsilon-1}\mathsf{poly}(\lambda)$, which approaches 1 for sufficiently large n, while having a key of sublinear size.

In order to declutter the presentation, we will switch from multiplicative notation of group operations in $\mathbb{G}$ to additive notion in the following discussion. That is we will denote group elements g^x by $[x]$, and write $\alpha \cdot [x]$ instead of $(g^x)^\alpha$. Furthermore, we will consider vectors and matrices of group elements, i.e. if $\mathbf{x} \in \mathbb{Z}_p^k$ is a vector, then $[\mathbf{x}]$ is its element-wise encoding in the group $\mathbb{G}$. Likewise, we write an encoding of a matrix $\mathbf{A} \in \mathbb{Z}_p^{k\times l}$ as $[\mathbf{A}]$.

In our discussion above we considered a HPS for the *two-dimensional* DDH language, i.e. the language consisting of all $r \cdot [\mathbf{v}]$ given two $[\mathbf{v}]$, where $\mathbf{v} \in \mathbb{Z}_p^2$ is a randomly chosen 2-dimensional vector over $\mathbb{Z}_p$.

Thus let $\mathbf{v} \in \mathbb{Z}_p^k$ be a randomly chosen k-dimensional vector. The goal of the k-dimensional DDH problem is to distinguish $([\mathbf{v}], t \cdot [\mathbf{v}])$ from $([\mathbf{v}], [\mathbf{u}])$, where $\mathbf{v}$ and $\mathbf{u}$ are chosen uniformly random from $\mathbb{Z}_p^k$ and r is chosen uniformly from $\mathbb{Z}_p$. It follows routinely via a standard rerandomization argument that the k-dimensional DDH problem is hard, given that the 2-dimensional DDH problem is hard.

We can construct an HPS for k-DDH analogously to the 2-dimensional case: Fix a vector $[\mathbf{v}] \in \mathbb{G}^k$. The secret key sk is a random vector $\boldsymbol{\alpha} \in \mathbb{Z}_p^k$, whereas the public key is given by $[\mathsf{pk}] = \boldsymbol{\alpha}^\top[\mathbf{v}]$, i.e. the inner product of $\boldsymbol{\alpha}$ and $[\mathbf{v}]$. Given a vector $[\mathbf{w}] = r \cdot [\mathbf{v}]$ and a witness r, the Encap algorithm computes $[\mathsf{K}] = r \cdot [\mathsf{pk}]$. On the other hand, given any vector $[\mathbf{w}] \in \mathbb{G}^k$ and a secret key $\mathsf{sk} = \boldsymbol{\alpha}$, the Decap algorithm computes $\boldsymbol{\alpha}^\top \cdot [\mathbf{w}]$. Arguing correctness and smoothness are again simple exercises in linear algebra. Furthermore, this HPS satisfies a stronger notion of $k-1$-smoothness: Given uniformly random $[\mathbf{w}_1], \ldots, [\mathbf{w}_{k-1}]$, it holds that

$$(\mathsf{pk}, \boldsymbol{\alpha}^\top[\mathbf{w}_1], \ldots, \boldsymbol{\alpha}^\top[\mathbf{w}_{k-1}]) \approx_s (\mathsf{pk}, [u_1], \ldots, [u_{k-1}]),$$

where the $[u_1], \ldots, [u_{k-1}]$ are uniformly random in $\mathbb{G}$. Establishing this is again routine linear algebra.

We will first briefly discuss how the HPS can be made programmable. In essence, we follow the same idea as above: We take a balance function $\mathsf{HC} : \mathbb{G} \to \{0,1\}$ and define the Decap algorithm to compute $\mathsf{HC}(\boldsymbol{\alpha}^\top[\mathbf{w}])$. We claim this construction is $k-1$-programmable. That is, given $[\mathbf{v}]$, $[\mathsf{pk}] = [t]$, uniformly random $[\mathbf{w}_1], \ldots [\mathbf{w}_{k-1}]$ together with the witnesses $\mathbf{v}$, t and $\mathbf{w}_1, \ldots, \mathbf{w}_{k-1}$, and a random $\mathsf{K} = (\mathsf{K}_1, \ldots, \mathsf{K}_{k-1}) \in \{0,1\}^{k-1}$, we can efficiently sample a uniformly random $\boldsymbol{\alpha} \in \mathbb{Z}_p^k$ such that $t = \boldsymbol{\alpha}^\top\mathbf{v}$ and $\mathsf{K}_i = \mathsf{HC}(\boldsymbol{\alpha}^\top[\mathbf{w}_i])$ for $i = 1, \ldots, n$. We proceed as above: First we choose uniformly random $z_i \in \mathbb{Z}_p$ such that $\mathsf{K}_i = \mathsf{HC}([z_i])$ for all i. Then we get the linear equation system

$$\begin{aligned}
\boldsymbol{\alpha}^\top\mathbf{v} &= t \\
\boldsymbol{\alpha}^\top\mathbf{w}_1 &= z_1 \\
&\vdots \\
\boldsymbol{\alpha}^\top\mathbf{w}_{k-1} &= z_{k-1}.
\end{aligned}$$

both $\mathcal{A}'_2$ and $\mathcal{A}'_3$. This is to *communicate* a potentially large public key PK from $\mathcal{A}_1$ to $\mathcal{A}_3$ without putting a burden on the leakage-budget of the leaker-stage $\mathcal{A}'_2$. One could consider an alternative definition where this communication from $\mathcal{A}'_1$ to $\mathcal{A}'_3$ is not allowed. In such a setting we could still prove our construction secure by *compressing* the auxiliary information aux from which PK and c_0 are generated using a PRG

Since the $\mathbf{w}_i$ are chosen uniformly random, this system has full rank w.o.p., and hence we can find a matching secret key $\boldsymbol{\alpha}$ via simple linear algebra.

Now, plugging this programmable HPS into our construction of incompressible PKE, we obtain the following parameters.

- A single public pk consisting of one group element can be used to encapsulate k key bits. Hence, to encapsulate n key bits we need n/k public keys amounting to n/k group elements.
- The ciphertext header now contains $k \cdot (k-1) \leq k^2$ group elements (in the above notation the vectors $[\mathbf{w}_1], \ldots, [\mathbf{w}_{k-1}]$).

Hence, if we want to strike a balance where the (additive) ciphertext overhead is of the same size as the public key, we obtain the relation

$$\frac{n}{k} = k^2,$$

which yields to $k = n^{1/3}$. Hence, for this choice of parameters the public key consists of a $n^{2/3}$ group elements (which is sublinear), and the size of the ciphertext is $n + n^{2/3}\log(|\mathbb{G}|) = n(1 - n^{-1/3}\log(|\mathbb{G}|))$ bits, which approaches rate 1.

2.5 Extension to CCA security

The scheme described so far achieves IND-CPA incompressible security. This work also considers an IND-CCA2 incompressible security definition where the adversary gets oracle access to a decryption oracle.

To achieve IND-CCA2 security, we follow the framework of [CS02]. We add a second hash proof system that acts as integrity proof for ciphertexts. The second hash proof system does not need to be programmable but universal$_2$ [CS02] or 2-smooth [ABP15]. It allows the decryption oracle to only answer queries to honestly generated ciphertexts. This mechanism ensures that the decryption oracle does not give up entropy of the programmable HPS's secret key.

In the main body of this work, we provide the full construction that achieves this level of security.

2.6 Incompressible Encryption in the ROM

Finally, we present a scheme that is secure in the ideal cipher model (proven to be equivalent to ROM) but insecure when we use a concrete hash function. The scheme is a simple hybrid encryption scheme where the symmetric encryption is one huge block cipher.

More concretely, the public key is composed of a pk of an IND-CPA scheme. To encrypt a message, we first sample two strings r, k of size λ and compute $c' \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{k})$. Then we compute $d = \mathsf{P}_\mathsf{k}((r, \mathsf{m}))$ where P is modelled as an ideal cipher oracle.

The scheme is secure in the ROM by observing that: i) The adversary cannot query the ideal cipher oracle with key k before receiving the secret key as it would break the IND-CPA security of the underlying PKE scheme; and ii) Given the secret key, the last stage adversary cannot query the ideal cipher oracle $\mathsf{P}_\mathsf{k}^{-1}$ on d because the limitations of the state size make sure d has high min-entropy. The adversary can also not query P_k on (r, m) as it would have to guess r. Therefore, the adversary has almost no information about the message.

However, if we use a fully-homomorphic encryption (FHE) scheme as a PKE and if we instantiate the ideal cipher oracle with a specific block cipher P, it is the scheme becomes breakable. The key idea is that as soon as the ideal cipher oracle is instantiated with a block cipher P, the adversary has access to the code of P and can thus run it homomorphically under the FHE. Concretely, the adversary chooses two messages m_0, m_1 and, after receiving the challenge encryption of m_b, it can unmask m_b inside the FHE and compare it with m_0, m_1. The resulting evaluated ciphertext contains a single bit which is much smaller than the original ciphertext. After receiving the secret key, it can decrypt b and break the IND-CPA incompressible security of the scheme.

3 Preliminaries

The acronym PPT denotes "probabilistic polynomial time". Throughout this work, λ denotes the security parameter. By $\mathsf{negl}(\lambda)$, we denote a negligible function in λ, that is, a function that vanishes faster than any inverse polynomial in λ. Let $n \in \mathbb{N}$. Then, $[n]$ denotes the set $\{1, \ldots, n\}$. If $\mathcal{A}$ is an algorithm, we denote by $y \leftarrow \mathcal{A}(x)$ the output y after running $\mathcal{A}$ on input x. If S is a (finite) set, we denote by $x \leftarrow\!\$\, S$ the experiment of sampling uniformly at random an element x from S. If D is a distribution over S, we denote by $x \leftarrow\!\$\, D$ the element x sampled from S according to D.

For two probability distributions X, Y, we use the notation $X \approx_s Y$ to state that the distributions are statistically indistinguishable and $X \approx_c Y$ to state that the distributions are computationally indistinguishable.

For ease of notation, in any of our constructions we assume public parameters p are known to every algorithm and every secret key sk also contains the corresponding public key pk.

We present some information-theoretical notions and results that will be instrumental throughout this work.

Definition 1 (Average Min-Entropy [DORS08])**.** *For two jointly distributed random variables (X, Y), the average min-entropy of X conditioned on Y is defined as*

$$\tilde{H}_\infty(X|Y) = -log(\mathbb{E}_{y \leftarrow\!\$\, Y}[max_x \Pr[X = x|Y = y]]).$$

Lemma 1 (Lemma 2.2 b) of [DORS08])**.** *For random variables X, Y, Z where Y is supported over a set of size T, we have*

$$\tilde{H}_\infty(X|(Y, Z)) \geq \tilde{H}_\infty((X, Y)|Z) - log(T) \geq \tilde{H}_\infty(X|Z) - log(T).$$

Definition 2 (Average-Case Extractor [DORS08])**.** *Let $n, d, m \in \mathbb{N}$. A function $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k, ϵ) strong average-case min-entropy extractor if, for all random variables (X, Y) where X takes values in $\{0,1\}^n$ and $\tilde{H}_\infty(X|Y) \geq k$, we have that $(U_d, \mathsf{Ext}(X, U_d), Y)$ is ϵ-close to (U_d, U_m, Y), where U_d and U_m are independent uniformly random strings of length d and m respectively.*

Lemma 2 (Generalized Leftover Hash Lemma 2.4 of [DORS08])**.** *Let $n, m \in \mathbb{N}$. Let $\{H_r : \{0,1\}^n \to \{0,1\}^m\}_{r \in R}$ be a family of universal hash functions, then $\mathsf{Ext}(x, r) \mapsto H_r(x)$ is an average-case (k, ϵ)-strong extractor whenever $m \leq k - 2log(\frac{1}{\epsilon}) + 2$.*

Definition 3 (Pseudorandom Generator)**.** *Let $n, m = \mathsf{poly}(\lambda)$. A function $G : \{0,1\}^n \to \{0,1\}^m$ is a pseudorandom generator if, for uniformly random $\mathsf{s} \leftarrow\!\$\, \{0,1\}^n$ and $r \leftarrow\!\$\, \{0,1\}^m$, we have*

$$G(\mathsf{s}) \approx_c r.$$

Definition 4 (Collision-Resitant Hash Function)**.** *Let $n, m, l = \mathsf{poly}(\lambda)$. A collision-resistant hash function is a seeded function $\mathsf{CRHF} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^l$ with the property that for all PPT adversaries $\mathcal{A}$, random seed $\mathsf{s} \in \{0,1\}^n$ we have $\mathcal{A}(\mathsf{s})$ outputs x,x' with $x \neq x'$ and $\mathsf{CRHF}_\mathsf{s}(x) = \mathsf{CRHF}_\mathsf{s}(x')$ with negligible probability.*

3.1 Decisional Diffie-Hellman assumption

In the following, let $\mathcal{G}$ be a (prime-order) *group generator*, that is, $\mathcal{G}$ is an algorithm that takes as an input a security parameter 1^λ and outputs $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is the description of a multiplicative cyclic group, p is the order of the group which is always a prime number unless differently specified, and g is a generator of the group. Sometimes we denote the size of the group by $|\mathbb{G}|$.

We denote by $[a]$ be value g^a. Similarly, if $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$ is a matrix with entries $a_{i,j}$ then $[\mathbf{A}]$ denotes the matrix where each (i, j)-entry is the value $g^{a_{i,j}}$. Note that given $\mathbf{x} \in \mathbb{Z}_p^n$, $\mathbf{y} \in \mathbb{Z}_p^m$ and $[\mathbf{A}]$, we can compute $\mathbf{x}^T[\mathbf{A}] = [\mathbf{x}^T\mathbf{A}]$ and $[\mathbf{A}]\mathbf{y} = [\mathbf{A}\mathbf{y}]$.

In the following we state the decisional version of the Diffie-Hellman (DDH) assumption.

Definition 5 (Decisional Diffie-Hellman Assumption)**.** *Let* $(\mathbb{G}, p, g) \leftarrow\!\!\$\, \mathcal{G}(1^\lambda)$*. We say that the DDH assumption holds (with respect to* $\mathcal{G}$*) if for any PPT adversary* $\mathcal{A}$

$$|\Pr[1 \leftarrow \mathcal{A}((\mathbb{G}, p, g), ([a], [b], [ab]))] - \Pr[1 \leftarrow \mathcal{A}((\mathbb{G}, p, g), ([a], [b], [c]))]| \leq \mathsf{negl}(\lambda)$$

where $a, b, c \leftarrow\!\!\$\, \mathbb{Z}_p$*.*

3.2 Public-Key Encryption

Definition 6 (Public-Key Encryption)**.** *A public-key encryption (PKE) scheme is a triple of PPT algorithms*

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$**:** *Given the security parameter* λ *the key-generation algorithm outputs a public key* pk *and a secret key* sk*.*

$c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$**:** *Given a public key* pk *and a message* m *encryption outputs a ciphertext* c*.*

$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$**:** *Given a secret key* sk *and a ciphertext* c *decryption outputs a message* m*.*

Correctness. For all $\lambda, S \in \mathbb{N}$, messages m and $(\mathsf{pk}, \mathsf{sk})$ in the range of KeyGen we have that $\mathsf{m} = \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{m}))$.

IND-CPA security. For all $\lambda \in \mathbb{N}$ and all adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have that

$$\mathsf{Pr}\left[b \leftarrow \mathcal{A}_2(\mathsf{st}, c) : \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pk}) \\ b \leftarrow\!\!\$\, \{0,1\} \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_b)\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

We now provide the definition of fully-homomorphic encryption.

Definition 7 (Fully-Homomorphic Encryption)**.** *A fully-homomorphic encryption (FHE) scheme is PKE scheme with the following additional algorithm:*

$c \leftarrow \mathsf{Eval}(\mathsf{pk}, C, (c_1, \ldots, c_\ell))$**:** *Given public key* pk*, a circuit* C *and ciphertexts* $(c_1, \ldots, c_\ell)$ *the evaluation algorithm outputs a new ciphertext* c*.*

IND-CPA is defined in a analogous way as for PKE. We now present the definitions of homomorphic correctness and compactness.

Homomorphic correctness. For all $\lambda, S \in \mathbb{N}$, messages m, any circuit $C : \{0,1\}^\ell \to \{0,1\}$ $(\mathsf{pk}, \mathsf{sk})$ in the range of KeyGen we have that

$$b = \mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, C, (\mathsf{Enc}(\mathsf{pk}, b_1), \ldots, \mathsf{Enc}(\mathsf{pk}, b_\ell))))$$

where $b \leftarrow C(b_1, \ldots, b_\ell)$.

Compactness. There exists a polynomial p such that for all $\lambda \in \mathbb{N}$, all circuits C, all inputs $b_1, \ldots, b_\ell$, all $(\mathsf{sk}, \mathsf{pk})$ in the support of $\mathsf{KeyGen}(1^\lambda)$, and all c_i in the support of $\mathsf{Enc}(\mathsf{pk}, b_i)$ it holds that $|\mathsf{Eval}(\mathsf{pk}, C, (c_1, ..., c_\ell))| = p(\lambda, |C(b_1, ..., b_\ell)|)$.

3.3 HILL-Entropic Encodings

We recall the notion of HILL-entropic encodings from [MW20].

Definition 8 (HILL-Entropic Encodings [MW20])**.** *An (α, β)-HILL-entropic encoding scheme with selective security in the CRS setting consists of two PTT algorithms:*

- $c \leftarrow \mathsf{En}_{\mathsf{crs}}(1^\lambda, \mathsf{m})$*: An encoding algorithm that takes a common random string* crs *and a message* m *producing an encoding* c*.*
- $m \leftarrow \mathsf{De}_{\mathsf{crs}}(c)$*: A decoding algorithm that takes a common random string* crs *and an encoding* c *and produces a message* m*.*

Correctness. There is some negligible μ such that for all $\lambda \in \mathbb{N}$ and all $m \in \{0,1\}^*$ we have

$$\Pr[\mathsf{De}_{\mathsf{crs}}(\mathsf{En}_{\mathsf{crs}}(1^\lambda, \mathsf{m})) = \mathsf{m}] = 1 - \mu(\lambda).$$

α-Expansion. For all $\lambda, k \in \mathbb{N}$ and all $\mathsf{m} \in \{0,1\}^k$ we have $|\mathsf{En}_{\mathsf{crs}}(1^\lambda, \mathsf{m})| \leq \alpha(\lambda, k)$.

β-HILL-Entropy. There exists an algorithm SimEn s.t. for any polynomial $k = k(\lambda)$ and any ensamble of messages $\mathsf{m} = \{\mathsf{m}_\lambda\}$ of length $|\mathsf{m}_\lambda| = k(\lambda)$, consider the following "real" experiment:

- $\mathsf{crs} \leftarrow\!\!\$\ \{0,1\}^{t(\lambda,k)}$
- $c \leftarrow \mathsf{En}_{\mathsf{crs}}(1^\lambda, \mathsf{m}_\lambda)$

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

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

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 $\beta(\lambda, k) = k(1 - o(1)) - \mathsf{poly}(\lambda)$. Moran and Wichs [MW20] provide good HILL-entropic encodings from DCR [Pai99, DJ01] or LWE [Reg05] in the CRS model. They also show that the CRS must be as big as the encoded message.

3.4 Random Oracle Model

In our constructions we use the ideal cipher model, which can be dated back to Shannon [Sha49] and proven to be indifferentiable from the random oracle model [BR93, HKT11].

For any $n \in \mathbb{N}$ the ideal cipher model provides oracle access a keyed permutation $\mathsf{P} : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$, which for each key k we have P_k is an independent random permutation.

4 Incompressible Symmetric-Key Encryption

In this section, we define incompressible symmetric-key encryption (SKE) and give a construction from entropic encodings.

4.1 Definition

First, we recall the notion of forward-secure storage [Dzi06b] under the name of incompressible symmetric-key encryption. For our purposes we only need IND-EAV style security but this could be extended similar to what we did with incompressible public-key encryption.

Definition 9 (Incompressible SKE)**.** *An incompressible symmetric-key encryption scheme is a tuple of PPT algorithms using uniformly random keys* k

$c \leftarrow \mathsf{Enc}(\mathsf{k}, \mathsf{m})$**:** *Given a symmetric key* k *and a message* m *encryption it outputs a ciphertext* c*.*

$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$**:** *Given a symmetric key* k *and a ciphertext* c *decryption it outputs a message* m*.*

We require size of message space, size of key space, and size of ciphertext space to be polynomials over the security parameter λ and the space bound S; that is, $n = n(\lambda, S)$, $k = k(\lambda, S)$, and $l = l(\lambda, S)$ respectively.

Correctness For all $\lambda, S \in \mathbb{N}$, messages m and keys $\mathsf{k} \in \{0,1\}^k$ we have that $\mathsf{m} = \mathsf{Dec}(\mathsf{k}, \mathsf{Enc}(\mathsf{k}, \mathsf{m}))$

Security For security parameter λ and space bound S, a symmetric-key encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ has incompressible SKE security if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ the probability of winning the following experiment is $\leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

$\mathsf{Dist}^{\mathsf{IncomSKE}}_{\mathcal{A},\Pi}(\lambda, S)$ **Experiment** :

- Run the adversary $(\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\lambda)$ to receive two messages m_0 and m_1
- Sample a bit $b \leftarrow\$ \{0,1\}$ uniformly at random
- Sample $\mathsf{k} \leftarrow\$ \{0,1\}^{n(\lambda,S)}$ uniformly at random
- Run $c \leftarrow \mathsf{Enc}(\mathsf{k}, \mathsf{m}_b)$ to encrypt m_b
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{k}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$
- The adversary wins if $b = b'$

4.2 Construction

Now we show how to build incompressible symmetric-key encryption using HILL-entropic encodings, extractors, and pseudorandom generators.

Construction 1. *Let* λ *be the security parameter,* S *be the space bound of the adversary and* n *be the size of the message space. Let* $(\mathsf{En}, \mathsf{De})$ *be an* (α, β)*-HILL-entropic encoding,* $\mathsf{Ext} : \{0,1\}^{\alpha(\lambda,n)} \times \{0,1\}^{d(\lambda)} \to \{0,1\}^\lambda$ *be a* $(\beta(\lambda, n) - S, \mathsf{negl}(\lambda))$ *strong average-case min-entropy extractor where* $d(\lambda)$ *is a polynomial and* $G : \{0,1\}^\lambda \to \{0,1\}^n$ *be a PRG.*

$\mathsf{Enc}(\mathsf{k}, \mathsf{m})$**:**

- *Parse* $\mathsf{k} = (\mathsf{k}_1, \mathsf{k}_2, \mathsf{crs})$*.*
- *Sample* $\mathsf{s} \leftarrow\$ \{0,1\}^\lambda$ *uniformly at random.*
- *Let* $c_1 \leftarrow \mathsf{En}_{\mathsf{crs}}(1^\lambda, G(\mathsf{s}) \oplus \mathsf{m})$*.*
- *Let* $c_2 \leftarrow \mathsf{s} \oplus \mathsf{Ext}(c_1, \mathsf{k}_1) \oplus \mathsf{k}_2$*.*
- *Return* $c = (c_1, c_2)$*.*

$\mathsf{Dec}(\mathsf{k}, c)$**:**

- *Parse* $\mathsf{k} = (\mathsf{k}_1, \mathsf{k}_2, \mathsf{crs})$.
- *Parse* $c = (c_1, c_2)$.
- *Let* $\mathsf{s} \leftarrow \mathsf{Ext}(c_1, \mathsf{k}_1) \oplus c_2 \oplus \mathsf{k}_2$.
- *Return* $\mathsf{De}_{\mathsf{crs}}(c_1) \oplus G(\mathsf{s})$.

Parameters. The ciphertexts are of size $\lambda + \alpha(\lambda, n)$. The keys are of size $d(\lambda) + t(\lambda, n)$, where $t(\lambda, n)$ is the size of the encoding's crs. Notice that the extractor exists if $\beta(\lambda, n) - S - 2\log\left(\frac{1}{\mathsf{negl}(\lambda)} + 2\right) \geq \lambda$ according to Lemma 2. So, the adversary is allowed a leakage of size $S \leq \beta(\lambda, n) - \lambda - 2\log\left(\frac{1}{\mathsf{negl}(\lambda)} + 2\right)$.

Therefore, if we choose a "good" entropic encoding we get a rate of $\frac{n}{n(1+o(1))+\mathsf{poly}(\lambda)}$, allowed leakage of $S = n(1 - o(1)) - \mathsf{poly}(\lambda)$, and keysize of $k = n(1 + o(1)) + \mathsf{poly}(\lambda)$.

Correctness. By the correctness of the entropic encoding $\mathsf{De}_{\mathsf{crs}}(\mathsf{En}_{\mathsf{crs}}(1^\lambda, G(\mathsf{s}) \oplus \mathsf{m})) = G(\mathsf{s}) \oplus \mathsf{m}$. Since Ext is deterministic under a fixed key k_1 then $\mathsf{Ext}(c_1, \mathsf{k}_1) \oplus c_2 \oplus \mathsf{k}_2 = \mathsf{Ext}(c_1, \mathsf{k}_1) \oplus \mathsf{s} \oplus \mathsf{Ext}(c_1, \mathsf{k}_1) = \mathsf{s}$. Therefore, $\mathsf{De}_{\mathsf{crs}}(c_1) \oplus G(\mathsf{s}) = \mathsf{m}$.

Theorem 2 (Security)**.** *The incompressible SKE presented in Construction 1 has incompressible SKE security if (*En,De*) is an* (α, β)*-HILL-entropic encoding,* Ext *is a* $(\beta(\lambda, n) - S, \mathsf{negl}(\lambda))$ *strong average-case min-entropy extractor, and* G *is a pseudorandom generator each with the listed parameters.*

Proof. We prove security via hybrids. First we list the hybrid and then argue their indistinguishability. In each hybrid we highlight the changes compared to the previous one.

H_0 :

- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(1^\lambda)$ to receive two messages m_0 and m_1.
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random.
- Sample $\mathsf{k} \leftarrow\$ \{0,1\}^n$ uniformly at random.
- Run $c \leftarrow \mathsf{Enc}(\mathsf{k}, \mathsf{m}_b)$ to encrypt m_b.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{k}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_1 we explicitly represent what happens in Enc.

H_1 :

- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(1^\lambda)$ to receive two messages m_0 and m_1.
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random.
- Sample $\mathsf{k}_1 \leftarrow\$ \{0,1\}^{d(\lambda,n)}$ uniformly at random.
- Sample $\mathsf{k}_2 \leftarrow\$ \{0,1\}^{\lambda}$ uniformly at random.
- Sample $\mathsf{crs} \leftarrow\$ \{0,1\}^{t(\lambda,n)}$ uniformly at random.
- Sample $\mathsf{s} \leftarrow\$ \{0,1\}^{\lambda}$ uniformly at random.
- Let $c_1 \leftarrow \mathsf{En}_{\mathsf{crs}}(1^\lambda, G(\mathsf{s}) \oplus \mathsf{m}_b)$.
- Let $c_2 \leftarrow \mathsf{s} \oplus \mathsf{Ext}(c_1, \mathsf{k}_1) \oplus \mathsf{k}_2$.

- Let $c \leftarrow (c_1, c_2)$ and $\mathsf{k} \leftarrow (\mathsf{k}_1, \mathsf{k}_2, \mathsf{crs})$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{k}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$

In H_2 we switch the entropic encoding to the simulated code that has a lot of entropy.

H_2 :

- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(1^\lambda)$ to receive two messages m_0 and m_1.
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random.
- Sample $\mathsf{k}_1 \leftarrow\$ \{0,1\}^{d(\lambda,n)}$ uniformly at random.
- Sample $\mathsf{k}_2 \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
-
- Sample $\mathsf{s} \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Let $(\mathsf{crs}, c_1) \leftarrow \mathsf{SimEn}(1^\lambda, G(\mathsf{s}) \oplus \mathsf{m}_b)$.
- Let $c_2 \leftarrow \mathsf{s} \oplus \mathsf{Ext}(c_1, \mathsf{k}_1) \oplus \mathsf{k}_2$.
- Let $c \leftarrow (c_1, c_2)$ and $\mathsf{k} \leftarrow (\mathsf{k}_1, \mathsf{k}_2, \mathsf{crs})$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{k}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

I H_3 we switch the order in which we sample c_2 and k_2.

H_3 :

- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(1^\lambda)$ to receive two messages m_0 and m_1.
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random.
- Sample $\mathsf{k}_1 \leftarrow\$ \{0,1\}^{d(\lambda,n)}$ uniformly at random.
-
- Sample $\mathsf{s} \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Let $(\mathsf{crs}, c_1) \leftarrow \mathsf{SimEn}(1^\lambda, G(\mathsf{s}) \oplus \mathsf{m}_b)$.
- Sample $c_2 \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Let $c \leftarrow (c_1, c_2)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S.
- Let $\mathsf{k}_2 \leftarrow c_2 \oplus \mathsf{Ext}(c_1, \mathsf{k}_1) \oplus \mathsf{s}$.
- Let $\mathsf{k} \leftarrow (\mathsf{k}_1, \mathsf{k}_2, \mathsf{crs})$.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{k}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_4 we replace the output of the extractor Ext by a uniformly random value.

H_4 :

- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(1^\lambda)$ to receive two messages m_0 and m_1.
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random.
- Sample $\mathsf{k}_1 \leftarrow\$ \{0,1\}^{d(\lambda,n)}$ uniformly at random.
- Sample $\mathsf{s} \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Let $(\mathsf{crs}, c_1) \leftarrow \mathsf{SimEn}(1^\lambda, G(\mathsf{s}) \oplus \mathsf{m}_b)$.
- Sample $c_2 \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Let $c \leftarrow (c_1, c_2)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S.
- Sample $\mathsf{k}_2 \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Let $\mathsf{k} \leftarrow (\mathsf{k}_1, \mathsf{k}_2, \mathsf{crs})$.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{k}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

Finally we replace the output of $G(\mathsf{s})$ by a uniformly random value.

H_5 :

- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(1^\lambda)$ to receive two messages m_0 and m_1.
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random.
- Sample $\mathsf{k}_1 \leftarrow\$ \{0,1\}^{d(\lambda,n)}$ uniformly at random.
- Sample $\mathsf{s} \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Sample $r \leftarrow\$ \{0,1\}^n$ uniformly at random.
- Let $(\mathsf{crs}, c_1) \leftarrow \mathsf{SimEn}(1^\lambda, r)$.
- Sample $c_2 \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Let $c \leftarrow (c_1, c_2)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S.
- Sample $\mathsf{k}_2 \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Let $\mathsf{k} \leftarrow (\mathsf{k}_1, \mathsf{k}_2, \mathsf{crs})$.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{k}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

$H_0 \approx H_1$:

The differences between H_0 and H_1 are purely syntactical. In H_1 we just show more detail of Enc.

$H_1 \approx_c H_2$:

Instead of sampling the common random string for the entropic encoding uniformly at random and then encoding $G(\mathsf{s}) \oplus \mathsf{m}$ we simulate both steps using SimEn. Assume there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ that can distinguish the two hybrids H_1 and H_2 with a non-negligible advantage of ϵ. From this we construct a PPT adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ that can break the β-HILL-entropy of $(\mathsf{En}, \mathsf{De})$ with advantage ϵ.

$\mathcal{A}'_1(1^\lambda)$:

- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(1^\lambda)$ to receive two messages m_0 and m_1
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random
- Sample $\mathsf{k}_1 \leftarrow\$ \{0,1\}^{d(\lambda,n)}$ uniformly at random
- Sample $\mathsf{s} \leftarrow\$ \{0,1\}^{\lambda}$ uniformly at random
- Return $G(\mathsf{s}) \oplus \mathsf{m}_b$

$\mathcal{A}'_2(\mathsf{crs}, c_1)$:

- Let $c_2 \leftarrow \mathsf{s} \oplus \mathsf{Ext}(c_1, \mathsf{k}_1)$
- Let $c \leftarrow (c_1, c_2)$
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{k}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$
- Return b'

If $\mathcal{A}$ can distinguish H_1 from H_2 then $\mathcal{A}'$ can distinguish a uniformly random $\mathsf{crs} \leftarrow\$ \{0,1\}^{t(\lambda,n)}$ and $c_1 \leftarrow \mathsf{En}(1^\lambda, G(\mathsf{s}) \oplus \mathsf{m}_b)$ from $(\mathsf{crs}, c_1) \leftarrow \mathsf{SimEn}(1^\lambda, G(\mathsf{s}) \oplus \mathsf{m}_b)$ as it perfectly simulates H_2 in the case that $(\mathsf{crs}, c_1) \leftarrow \mathsf{SimEn}(1^\lambda, G(\mathsf{s}) \oplus \mathsf{m}_b)$ and perfectly simulates H_1 in the other case.

$H_2 \approx H_3$:

In H_3 we switch the order in which we sample c_2 and k_2. From the view of the adversary this is statistically identical.

$H_3 \approx_s H_4$:

Let C_1, C_2, CRS, K_1, K_2, and ST_2 denote the random variables for the corresponding values in the experiment and U_λ independent uniform randomness of length λ. By the β-HILL entropy of the entropic encoding we know that $\tilde{H}_\infty(C_1|CRS) \geq \beta$. Using Lemma 1 we deduce that $\tilde{H}_\infty(C_1|(CRS, K_2, ST_2, C_2)) \geq \beta - 2\lambda - log(S)$. Therefore, the extractor gives us that $(K_1, K_2, CRS, ST_2, U_\lambda)$ is statistically close to $(K_1, K_2, CRS, ST_2, \mathsf{Ext}(C_1, K_1))$ which is exactly the view of $\mathcal{A}_3$.

$H_3 \approx_c H_4$:

In H_4 we encode a uniformly random string instead of $G(\mathsf{s}) \oplus \mathsf{m}_b$. Assume there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ that can distinguish the two hybrids H_3 and H_4 with a non-negligible advantage of ϵ. From this we construct a PPT adversary $\mathcal{A}'$ that can break the pseudorandomness of G with advantage ϵ.

$\mathcal{A}'(r')$:

- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(1^\lambda)$ to receive two messages m_0 and m_1
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random
- Sample $\mathsf{k}_1 \leftarrow\$ \{0,1\}^{d(\lambda,n)}$ uniformly at random
- Sample $\mathsf{s} \leftarrow\$ \{0,1\}^{\lambda}$ uniformly at random
- Sample $r \leftarrow r' \oplus \mathsf{m}_b$ uniformly at random
- Let $(\mathsf{crs}, c_1) \leftarrow \mathsf{SimEn}(1^\lambda, r)$
- Sample $c_2 \leftarrow\$ \{0,1\}^{\lambda}$ uniformly at random
- Let $c \leftarrow (c_1, c_2)$
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{k}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$
- Return b'

If $\mathcal{A}$ can distinguish H_3 from H_4 then $\mathcal{A}'$ can distinguish $G(\mathsf{s})$ with uniformly random $\mathsf{s} \leftarrow\$ \{0,1\}^{\lambda}$ from uniformly random $r' \leftarrow\$ \{0,1\}^n$ as it perfectly simulates H_3 in the case that $r' \leftarrow G(\mathsf{s})$ and perfectly simulates H_4 in the other case.

H_4 :

In H_4 the winning probability of the adversary is $\frac{1}{2}$ as it gets no information about b at all.

□

5 Programmable Hash Proof Systems

In this work we think of a hash proof systems as a key encapsulation mechanism where the encapsulated key is independent of the public key and the ciphertext under certain conditions. This allows us to later resample the secret key in the incompressibility experiments.

For our construction we need two different hash proof systems. One that is Y-programmable and one that is 2-smooth both using the same language.

5.1 Definitions

First we define hash proof system that we will use as a mask in our encryption scheme.

Definition 10 (Y-Programmable Hash Proof System [CS02, Kal05])**.** *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 a corresponding trapdoor $\mathsf{td}_{\mathcal{L}}$, an $x \in \mathcal{L}$ with its witness w and an $x \in Y$ with a corresponding trapdoor td_x*

- $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^k)$*: Given the security parameter λ, the encapsulated key size k the language generation algorithm that outputs public parameters p defining a language $\mathcal{L}$ and a trapdoor $\mathsf{td}_{\mathcal{L}}$ to that language.*
- $(x \in \mathcal{L}, w) \leftarrow \mathsf{samp}\mathcal{L}(\mathsf{p})$*: Given the public parameters, it outputs an element $x \in \mathcal{L}$ with the corresponding witness w.*
- $(x \in Y, \mathsf{td}_x) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$*: Given the public parameters 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:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{p})$*: Given the public parameters, the key generation algorithm outputs a public key pk and a secret key sk.*
- $\mathsf{k} \leftarrow \mathsf{Encap}(\mathsf{pk}, x, w)$*: Given the public lye pk, en element x and a witness w. the key encapsulation algorithm outputs an encapsulated key k.*
- $\mathsf{k} \leftarrow \mathsf{Decap}(\mathsf{sk}, x)$*: Given the secret key sk and any $x \in X$, the key decapsulation algorithm outputs an encapsulated key. k. Notice x can be outside $\mathcal{L}$.*
- $\mathsf{sk}' \leftarrow \mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}, x, \mathsf{k})$ *Given two trapdoors $\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x$, a secret key sk, an element $x \in Y$, and an encapsulated key k, the programming algorithm 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 $\lambda, k \in \mathbb{N}$ if we sample $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^k)$, $\mathcal{L} \ni x \leftarrow \mathsf{samp}\mathcal{L}(\mathsf{p})$, and $(x^* \in Y, \mathsf{td}_{x^*}) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$, we have the computational indistinguishability: $x \approx_c x^*$.

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

Notice, if $Y = X \setminus \mathcal{L}$ then Y-programmable smoothness implies smoothness.

Next we recall 2-smooth hash proof systems with our adjusted notation.

Definition 11 (2-Smooth Hash Proof System [CS02, ABP15])**.** *A 2-smooth hash proof system is defined over a NP language $\mathcal{L} \subset X$ as above The hash proof system itself consists of the following algorithms:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{p})$*: Given the public parameters, the key generation algorithm that outputs a public key* pk *and a secret key* sk*.*
- $\mathsf{k} \leftarrow \mathsf{Encap}(\mathsf{pk}, x, w, \tau)$*: Given public key* pk*, an element of the language $x \in \mathcal{L}$, its witness w, and a tag τ, the key encapsulation algorithm outputs an encapsulated key* k*.*
- $\mathsf{k} \leftarrow \mathsf{Decap}(\mathsf{sk}, x, \tau)$*: Given the secret key* sk*, any $x \in X$, and a tag τ. the key decapsulation algorithm outputs an encapsulated key* k*. Notice x can be outside $\mathcal{L}$.*

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 $\mathsf{KeyGen}(\mathsf{p})$, $x \in \mathcal{L}$, tags τ, and for $\mathsf{k} \leftarrow \mathsf{Encap}(\mathsf{pk}, x, w, \tau)$, we have $\mathsf{k} = \mathsf{Decap}(\mathsf{sk}, x, \tau)$ with $|\mathsf{k}| = k$.

Language Indistinguishability. Exactly as above.

2-Smoothness. For all $\lambda, k \in \mathbb{N}$, $(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$ in the range of $\mathsf{Gen}(1^\lambda, 1^k)$, $x, x' \in X \setminus \mathcal{L}$, two tags τ, τ' such that $(x, \tau) \neq (x', \tau')$, let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{p})$ and sample $\mathsf{k} \leftarrow\!\!\$\, \{0,1\}^k$ we have computational indistinguishability between $(\mathsf{pk}, \mathsf{Decap}(\mathsf{sk}, x, \tau), \mathsf{Decap}(\mathsf{sk}, x', \tau'))$ and $(\mathsf{pk}, \mathsf{Decap}(\mathsf{sk}, x, \tau), \mathsf{k})$.

5.2 Programmable Hash Proof System from DDH

In our protocols we need programmable HPS with a big encapsulated key space (for classic notation [CS02] this would be called the hash space).

Some smooth hash proof systems are easily transformed into programmable HPS with big encapsulated keys by generating more public keys and using them on the same $x \in X$. These HPS include the one from weak pseudorandom effective group actions [ADMP20]. That transformation causes the public key size to scale linearly with the size of the encapsulated key and leave the size of the ciphertext indepent of the encapsulated key size.

We present a variant of the original [CS02] HPS with an interesting trade off. Here both public key size and ciphertext size scale in the 2/3-power with k, the size of the encapsulated key.

Construction 2. *Let $\mathsf{HC} : \mathbb{G} \times \{0,1\}^{log(|\mathbb{G}|)} \to \{0,1\}$ denote a 1-bit randomness extractor over a group element; if this function is applied over a matrix of group elements, then it means that the function is applied entry-wise with the same randomness. In the following let $\ell, s \in \mathbb{N}$ such that $\ell \cdot s = k$. We get an interesting tradeoff for our application when $\ell = k^{1/3}$ and $s = k^{2/3}$.*

$\mathsf{Gen}(1^\lambda, 1^k)$:

- $(\mathbb{G}, p, g) \leftarrow\!\!\$\, \mathcal{G}(1^\lambda)$.
- *Sample* $\mathbf{h} \leftarrow\!\!\$\, \mathbb{Z}_p^\ell \setminus \{\mathbf{0}\}$ *uniformly at random.*
- *Return* $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}])$ *and* $\mathsf{td}_{\mathcal{L}} = \mathbf{h}$.

$\mathsf{samp}\mathcal{L}(\mathsf{p})$:

- *Parse* $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}])$.

- *Sample* $\mathbf{y} \leftarrow\$ \mathbb{Z}_p^{\ell-1}$ *uniformly at random.*
- *Return* $x = [\mathbf{y}]$ *and* $w = \mathbf{y}$.

$\mathsf{samp}Y(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$:

- *Parse* $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}])$.
- *Let* $\mathsf{td}_{\mathcal{L}} = \mathbf{h}$.
- *Sample* $\mathbf{E} \leftarrow\$ \mathbb{Z}_p^{\ell \times (\ell-1)}$ *such that* $\begin{pmatrix}\mathbf{h} & \mathbf{E}\end{pmatrix}$ *is invertible uniformly at random.*
- *Return* $x = [\mathbf{E}]$ *and* $w = \mathbf{E}$.

$\mathsf{KeyGen}(\mathsf{p})$**:**

- *Parse* $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}])$.
- *Sample* $r \leftarrow\$ \{0,1\}^{log(|\mathbb{G}|)}$ *the public randomness for a extractor.*
- *Sample* $\mathbf{A} \leftarrow\$ \mathbb{Z}_p^{s \times \ell}$ *uniformly at random.*
- *Return* $\mathsf{pk} = (\mathbf{A}[\mathbf{h}], r)$ *and* $\mathsf{sk} = \mathbf{A}$.

$\mathsf{Encap}\ \ (\mathsf{pk}, c = [\mathbf{h}\mathbf{y}^t], w = \mathbf{y})$*:*

- *Parse* $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}] \in \mathbb{G}^{\ell})$.
- *Parse* $\mathsf{pk} = ([\mathbf{f}] \in \mathbb{G}^s, r)$.
- *Let* $\mathbf{K} \leftarrow \mathsf{HC}([\mathbf{f}]\mathbf{y}^t, r)$ *the component-wise extractor of the outer product between* $\mathbf{f}$ *and* $\mathbf{y}$.
- *Return* $\mathsf{k} = \mathbf{K}$.

$\mathsf{Decap}\ \ (\mathsf{sk}, x = [\mathbf{E}] \in \mathbb{G}^{\ell \times (\ell-1)})$*:*

- *Parse* $\mathsf{pk} = ([\mathbf{f}], r)$
- *Parse* $\mathsf{sk} = \mathbf{A} \in \mathbb{Z}_p^{s \times \ell}$.
- *Let* $\mathbf{K} \leftarrow \mathsf{HC}(\mathbf{A}[\mathbf{E}], r)$ *the component-wise extractor of the product between* $\mathbf{A}$ *and* $[\mathbf{E}]$.
- *Return* $\mathsf{k} = \mathbf{K}$.

$\mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}, x, \mathsf{k})$**:**

- *Parse* $\mathsf{pk} = ([\mathbf{f}], r)$, $\mathsf{td}_{\mathcal{L}} = \mathbf{h} \in \mathbb{Z}_p^{\ell}$, $\mathsf{td}_x = \mathbf{E} \in \mathbb{Z}_p^{\ell \times (\ell-1)}$, $\mathsf{sk} = \mathbf{A}$, *and* $\mathsf{k} = \mathbf{K} \in \{0,1\}^{s \times (\ell-1)}$.
- *For each* $i \in [\ell-1], j \in [s]$ *sample* $B_{i,j} \leftarrow\$ \mathbb{Z}_p$ *such that* $K_{i,j} = \mathsf{HC}([B_{i,j}], r)$ *via rejection sampling.*
- *Set* $\mathbf{B} = (B)_{i,j}$. *Let* $\mathbf{A}' \leftarrow \begin{pmatrix}\mathbf{A}\mathbf{h} & \mathbf{B}\end{pmatrix}\begin{pmatrix}\mathbf{h} & \mathbf{E}\end{pmatrix}^{-1}$.
- *Return* $\mathsf{sk}' = \mathbf{A}'$.

Correctness. For any $(\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}]), \mathsf{td}_{\mathcal{L}})$ in the range of Gen, $(\mathsf{pk} = ([\mathbf{A}\mathbf{h}], r), \mathsf{sk} = \mathbf{A})$ in the range of KeyGen, and $[\mathbf{h}\mathbf{y}^t] \in \mathcal{L}$ we have $\mathsf{Encap}(\mathsf{pk}, [\mathbf{h}\mathbf{y}^t])$ outputs $\mathsf{k} = \mathsf{HC}([\mathbf{A}\mathbf{h}]\mathbf{y}^{\mathbf{t}}, r) = \mathsf{HC}([\mathbf{A}\mathbf{h}\mathbf{y}^t], r)$. Decapsulation then outputs $\mathsf{k} = \mathsf{HC}(\mathbf{A}[\mathbf{h}\mathbf{y}^t], r) = \mathsf{HC}([\mathbf{A}\mathbf{h}\mathbf{y}^t], r)$.

Programmability. Since we choose $\mathbf{h}$ and $\mathbf{E}$ s.t. $\begin{pmatrix}\mathbf{h} & \mathbf{E}\end{pmatrix}$ is invertible $\mathsf{Program}$ always outputs a matrix $\mathbf{A}'$ with the property that $\mathbf{A}'\mathbf{E} = \mathbf{B}$ and $\mathsf{k} = \mathsf{HC}([\mathbf{B}], r)$.

Programmable Smoothness. If we first sample k uniformly random and then program for the key k $\mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}, x, \mathsf{k})$ the resulting distribution over $\mathbf{B}$ will be uniformly random. And because $\begin{pmatrix}\mathbf{h} & \mathbf{E}\end{pmatrix}$ is invertible then $\mathbf{A}'$ is a uniformly random under the condition that $\mathbf{A}'\mathbf{h} = \mathbf{A}\mathbf{h}$. The same holds for $\mathbf{A}$. Therefore, $(\mathsf{pk}, \mathsf{sk} = \mathbf{A}, x)$ and $(\mathsf{pk}, \mathsf{sk}' = \mathbf{A}', x)$ are identically distributed.

Theorem 3 (Language Indistinguishability)**.** *If DDH is hard for $\mathcal{G}$ then elements from $\mathcal{L} = \{[\mathbf{h}]\mathbf{y}^t | \mathbf{y} \in \mathbb{Z}_p^{\ell-1}\}$ and $Y = \{[\mathbf{E}] | \mathbf{E} \in \mathbb{Z}_p^{\ell \times (\ell-1)} \wedge \begin{pmatrix}\mathbf{h} & \mathbf{E}\end{pmatrix}$ is invertible$\}$ of construction 2 are indistinguishable.*

Proof. We prove security via hybrids. First we list the hybrids and then argue their indistinguishability. In each hybrid we highlight the changes compared to the previous one.

H_0 :

- Let $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^k)$.
- Let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{p})$.
- Let $(x, w) \leftarrow \mathsf{samp}\mathcal{L}(\mathsf{p})$.
- Let $\mathsf{k} \leftarrow \mathsf{Encap}(\mathsf{pk}, x, w)$.
- Run the adversary $\mathcal{A}(\mathsf{pk}, \mathsf{sk}, x)$.

H_1 :

- Sample a group $(\mathbb{G}, p, g) \leftarrow\!\!\$\, \mathcal{G}(1^\lambda)$.
- Sample $r \leftarrow\!\!\$\, \{0,1\}^{log(|\mathbb{G}|)}$ the randomness for the extractor.
- Sample $\mathbf{h} \leftarrow\!\!\$\, \mathbb{Z}_p^\ell \setminus \{\mathbf{0}\}$ uniformly at random.
- Sample $\mathbf{A} \leftarrow\!\!\$\, \mathbb{Z}_p^{s \times \ell}$ uniformly at random.
- Let $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}])$.
- Let $\mathsf{pk} = ([\mathbf{A}\mathbf{h}], r)$ and $\mathsf{sk} = \mathbf{A}$.
- Sample $\mathbf{y} \leftarrow\!\!\$\, \mathbb{Z}_p^{\ell-1}$ uniformly at random.
- Let $x = [\mathbf{h}\mathbf{y}^t] = [\mathbf{C}]$.
- Run the adversary $\mathcal{A}(\mathsf{pk}, \mathsf{sk}, x)$.

$H_{2,i}$:

- Sample a group $(\mathbb{G}, p, g) \leftarrow\!\!\$\, \mathcal{G}(1^\lambda)$.
- Sample $r \leftarrow\!\!\$\, \{0,1\}^{\log(|\mathbb{G}|)}$ the randomness for the extractor.
- Sample $\mathbf{h} \leftarrow\!\!\$\, \mathbb{Z}_p^\ell \setminus \{\mathbf{0}\}$ uniformly at random.
- Sample $\mathbf{A} \leftarrow\!\!\$\, \mathbb{Z}_p^{s \times \ell}$ uniformly at random.
- Let $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}])$.
- Let $\mathsf{pk} = ([\mathbf{A}\mathbf{h}], r)$ and $\mathsf{sk} = \mathbf{A}$.
- Sample $\mathbf{y} \leftarrow\!\!\$\, \mathbb{Z}_p^{\ell-1}$ uniformly at random.
- Let $[\mathbf{C}] = [\mathbf{h}\mathbf{y}^t]$.
- Sample $\mathbf{E} \leftarrow\!\!\$\, \mathbb{Z}_p^{l \times (l-1)}$ uniformly at random.

- Replace the first i entries of $[\mathbf{C}]$ by the first i entries in $[\mathbf{E}]$.
- Let $x = [\mathbf{C}]$.
- Run the adversary $\mathcal{A}(\mathsf{pk}, \mathsf{sk}, x)$.

H_3 :

- Sample a group $(\mathbb{G}, p, g) \leftarrow\!\$\, \mathcal{G}(1^\lambda)$.
- Sample $r \leftarrow\!\$\, \{0,1\}^{\log(|\mathbb{G}|)}$ the randomness for the extractor.
- Sample $\mathbf{h} \leftarrow\!\$\, \mathbb{Z}_p^\ell \setminus \{\mathbf{0}\}$ uniformly at random.
- Sample $\mathbf{A} \leftarrow\!\$\, \mathbb{Z}_p^{s\times\ell}$ uniformly at random.
- Let $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}])$.
- Let $\mathsf{pk} = ([\mathbf{Ah}], r)$ and $\mathsf{sk} = \mathbf{A}$.
- Sample $\mathbf{E} \leftarrow\!\$\, \mathbb{Z}_p^{l\times(l-1)}$ uniformly at random such that $\begin{pmatrix}\mathbf{h} & \mathbf{E}\end{pmatrix}$ is invertible.
- Let $x = [\mathbf{E}]$.
- Run the adversary $\mathcal{A}(\mathsf{pk}, \mathsf{sk}, x)$.

$H_0 \approx H_1$:
The differences between H_0 and H_1 are purely syntactical. In H_1 we just show more detail of Gen and Encap.

$H_1 \approx H_{2,0}$:
The differences between H_1 and $H_{2,0}$ are purely syntactical.

$H_{2,i} \approx_c H_{2,i+1}$:
In $H_{2,i+1}$ we replace the $n+1$st element of $\mathbf{C}$ by a random one. Assume there exists a PPT adversary $\mathcal{A}$ that can distinguish the two hybrids $H_{2,i}$ and $H_{2,i+1}$ with a non-negligible advantage of ϵ. From this we construct a PPT adversary $\mathcal{A}'$ that can break DDH with advantage ϵ.

$\mathcal{A}'$ $((\mathbb{G}, p, g), ([a], [b], [\rho]))$:

- Let $u \leftarrow i \mod l$
- Let $v \leftarrow \lfloor i/l \rfloor$
- Sample $r \leftarrow\!\$\, \{0,1\}^{\log(|\mathbb{G}|)}$ the randomness for the extractor
- Sample $\mathbf{h} \leftarrow\!\$\, \mathbb{Z}_p^\ell \setminus \{\mathbf{0}\}$ uniformly at random
- Replace $[x_u]$ by $[a]$
- Sample $\mathbf{A} \leftarrow\!\$\, \mathbb{Z}_p^{s\times\ell}$ uniformly at random
- Let $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}])$
- Let $\mathsf{pk} = ([\mathbf{Ah}], r)$ and $\mathsf{sk} = \mathbf{A}$
- Sample $\mathbf{y} \leftarrow\!\$\, \mathbb{Z}_p^{\ell-1}$ uniformly at random
- For $u' \in [l]$ and $v' \in [l-1]$ let $C_{u',v'} \leftarrow \begin{cases} [\rho] & \text{if } u' = u, v' = v \\ [b]x_{u'} & \text{if } u' \neq u, v' = v \\ [x_{u'}]y_{v'} & \text{else} \end{cases}$
- Sample $\mathbf{E} \leftarrow\!\$\, \mathbb{Z}_p^{l\times(l-1)}$
- Replace the first i entries of $[\mathbf{C}]$ by the first i entries in $[\mathbf{E}]$
- Let $x = [\mathbf{C}]$
- Run the adversary $b' \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{sk}, x)$
- Return b'

If $\mathcal{A}$ distinguishes between $H_{2,i}$ and $H_{2,i+1}$ then $\mathcal{A}'$ distinguishes between $\rho = ab$ and ρ being uniformly random as $\mathcal{A}'$ perfectly simulates $H_{2,i}$ in the case that $\rho = ab$ and $H_{2,i+1}$ if r is uniformly random.

$H_{2,m} \approx_s H_3$:
$H_{2,m}$ is statistically close to H_3 because with probability $1 - \mathsf{negl}(\lambda)$ we have $\begin{pmatrix}\mathbf{h} & \mathbf{E}\end{pmatrix}$ is invertible.

□

Parameters. For an encapsulated key of size k this scheme roughly gets us public parameters of size $k^{1/3} \cdot \mathsf{poly}(\lambda)$, public key of size $k^{2/3} \cdot \mathsf{poly}(\lambda)$ and elements from X of size $k^{2/3} \cdot \mathsf{poly}(\lambda)$.

5.3 2-Smooth Hash Proof System from DDH

The above hash proof system only is programmable if $x \in Y$. To make our encryption scheme CCA secure we need a efficient way to check whether $x \in \mathcal{L}$ or $x \in X \setminus \mathcal{L}$. To do this we construct the 2-smooth hash proof system below that is defined over the same language.

Construction 3. *We construct a 2-smooth hash proof system with a output size of λ using an extractor $\mathsf{Ext} : \mathbb{G}^{\ell-1} \times \{0,1\}^p \to \{0,1\}^\lambda$ and a collision resistant hash function CRHF that maps into $\mathbb{Z}_p$. As a language description we use the same as in Construction 2.*

$\mathsf{KeyGen}(\mathsf{p})$**:**

- *Parse* $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}])$.
- *Sample* $r \leftarrow\!\!\$\, \{0,1\}^{log(|\mathbb{G}|)}$ *uniformly at random.*
- *Sample* $\mathsf{s} \leftarrow\!\!\$\, \{0,1\}^\lambda$ *uniformly at random.*
- *Sample* $\mathbf{a}, \mathbf{b} \leftarrow\!\!\$\, \mathbb{Z}_p^\ell$ *uniformly at random.*
- *Return* $\mathsf{pk} = (\mathbf{a}^t[\mathbf{h}], \mathbf{b}^t[\mathbf{h}], r, \mathsf{s})$ *and* $\mathsf{sk} = (\mathbf{a}, \mathbf{b})$.

Encap $(\mathsf{pk}, x = [\mathbf{h}\mathbf{y}^t] \in \mathbb{G}^{\ell \times (\ell-1)}, w = \mathbf{y} \in \mathbb{Z}_p^{\ell-1}, \tau)$*:*

- *Parse* $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}] \in \mathbb{G}^\ell)$ *and* $\mathsf{pk} = ([f], [f'] \in \mathbb{G}, r, \mathsf{s})$.
- *Let* $[\mathbf{d}] = ([f]\mathbf{y}) + (\mathsf{CRHF}_\mathsf{s}(x, \tau)[f']\mathbf{y})$.
- *Return* $\mathsf{k} = \mathsf{Ext}([\mathbf{d}^t], r)$.

Decap $(\mathsf{sk}, x = [\mathbf{E}] \in \mathbb{G}^{\ell \times (\ell-1)}, \tau)$*:*

- *Parse* $\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}] \in \mathbb{G}^\ell)$ *and* $\mathsf{pk} = ([f], [f'] \in \mathbb{G}, r, \mathsf{s})$.
- *Parse* $\mathsf{sk} = (\mathbf{a} \in \mathbb{Z}_p^\ell, \mathbf{b} \in \mathbb{Z}_p^\ell)$.
- *Parse* $x = [\mathbf{E}] \in \mathbb{G}^{\ell \times (\ell-1)}$.
- *Return* $\mathsf{k} = \mathsf{Ext}(\mathbf{a}^t[\mathbf{E}] + \mathsf{CRHF}_\mathsf{s}(x, \tau)\mathbf{b}^t[\mathbf{E}], r)$.

Correctness. For any $(\mathsf{p} = (\mathbb{G}, p, g, [\mathbf{h}]), \mathsf{td}_\mathcal{L})$ in the range of Gen, $(\mathsf{pk} = (\mathbf{a}^t[\mathbf{h}], \mathbf{b}^t[\mathbf{h}], r, \mathsf{s}), \mathsf{sk} = (\mathbf{a}, \mathbf{b}))$ in the range of KeyGen, and $[\mathbf{h}\mathbf{y}^t] \in \mathcal{L}$ we have $\mathsf{Encap}(\mathsf{pk}, [\mathbf{h}\mathbf{y}^t])$ outputs

$$\mathsf{k} = \mathsf{Ext}\left(\left(([f]\mathbf{y}) + ([f]\mathsf{CRHF}_\mathsf{s}(x, \tau)\mathbf{y})\right)^t, r\right) = \mathsf{Ext}\left([(\mathbf{a}^t\mathbf{h})\mathbf{y}^t + \mathsf{CRHF}_\mathsf{s}(x, \tau)(\mathbf{b}^t\mathbf{h})\mathbf{y}^t], r\right).$$

On the other hand, decapsulation outputs

$$\mathsf{k} = \mathsf{Ext}(\mathbf{a}^t[\mathbf{h}\mathbf{y}^t] + \mathsf{CRHF}_\mathsf{s}(x, \tau)\mathbf{b}^t[\mathbf{h}\mathbf{y}^t], r) = \mathsf{Ext}\left([(\mathbf{a}^t\mathbf{h})\mathbf{y}^t + \mathsf{CRHF}_\mathsf{s}(x, \tau)(\mathbf{b}^t\mathbf{h})\mathbf{y}^t], r\right)$$

Language Indistinguishability. Since we use the same language as in construction 2 the language indistinguishability holds by the same argument.

2-Smoothness. For all $\lambda, n \in \mathbb{N}$, $(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$ in the range of $\mathsf{Gen}(\mathsf{p})$, $x, x' \in X \setminus \mathcal{L}$, two tags τ, τ' such that $(x, \tau) \neq (x', \tau')$, let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{p})$ and sample $\mathsf{k} \leftarrow\!\!\$\, \{0,1\}^m$. Let $\gamma \leftarrow \mathsf{CRHF}_{\mathsf{s}}(x, \tau)$ and $\gamma' \leftarrow \mathsf{CRHF}_{\mathsf{s}}(x', \tau')$. Using $(x, \tau) \neq (x', \tau')$ and the collision resistance of CRHF we can assume that $\gamma \neq \gamma'$.

In the following let $\mathbf{d} = \mathbf{a}^t[\mathbf{E}] + \gamma \mathbf{b}^t[\mathbf{E}]$ (computed in $\mathsf{Decap}(\mathsf{sk}, x, \tau)$) and $\mathbf{d'} = \mathbf{a}^t[\mathbf{E}'] + \gamma' \mathbf{b}^t[\mathbf{E}']$ (computed in $\mathsf{Decap}(\mathsf{sk}, x', \tau')$). Then the following equation holds:

$$\begin{pmatrix} f & f' & \mathbf{d}^t & \mathbf{d'}^t \end{pmatrix} = \begin{pmatrix} \mathbf{a}^t & \mathbf{b}^t \end{pmatrix} \begin{pmatrix} \mathbf{h} & \mathbf{0} & \mathbf{E} & \mathbf{E}' \\ \mathbf{0} & \mathbf{h} & \gamma \mathbf{E} & \gamma' \mathbf{E}' \end{pmatrix}$$

If $x, x' \in X \setminus \mathcal{L}$ then there exists a column $\mathbf{z}$ with index i in $\mathbf{E}$ s.t. $\mathbf{z}$ is linearly independent of $\mathbf{h}$ and $\mathbf{z}'$ with index i' in $\mathbf{E}'$ s.t. $\mathbf{z}'$ is l.i. of $\mathbf{h}$. Then the following equation also holds:

$$\begin{pmatrix} f & f' & d_i & d'_{i'} \end{pmatrix} = \begin{pmatrix} \mathbf{a}^t & \mathbf{b}^t \end{pmatrix} \begin{pmatrix} \mathbf{h} & \mathbf{0} & \mathbf{z} & \mathbf{z}' \\ \mathbf{0} & \mathbf{h} & \gamma \mathbf{z} & \gamma' \mathbf{z}' \end{pmatrix}$$

Now, we argue that the matrix on the right side has rank 4. We have that $\begin{pmatrix} \mathbf{h} \\ \mathbf{0} \end{pmatrix}$ and $\begin{pmatrix} \mathbf{0} \\ \mathbf{h} \end{pmatrix}$ are linearly independent. Moreover, $\begin{pmatrix} \mathbf{z} \\ \gamma \mathbf{z} \end{pmatrix}$ is outside the span of $\begin{pmatrix} \mathbf{h} \\ \mathbf{0} \end{pmatrix}$ and $\begin{pmatrix} \mathbf{0} \\ \mathbf{h} \end{pmatrix}$ because $\mathbf{h}$ and $\mathbf{z}$ are linearly independent. Finally, $\begin{pmatrix} \mathbf{z}' \\ \gamma' \mathbf{z}' \end{pmatrix}$ is outside the span of $\begin{pmatrix} \mathbf{z} \\ \gamma \mathbf{z} \end{pmatrix}$, $\begin{pmatrix} \mathbf{h} \\ \mathbf{0} \end{pmatrix}$, and $\begin{pmatrix} \mathbf{0} \\ \mathbf{h} \end{pmatrix}$. To see this, assume that this is not the case, i.e., that there exists a linear combination

$$\begin{pmatrix} \mathbf{z}' \\ \gamma' \mathbf{z}' \end{pmatrix} = c_1 \begin{pmatrix} \mathbf{z} \\ \gamma \mathbf{z} \end{pmatrix} + c_2 \begin{pmatrix} \mathbf{h} \\ \mathbf{0} \end{pmatrix} + c_3 \begin{pmatrix} \mathbf{0} \\ \mathbf{h} \end{pmatrix}.$$

Assume there exist c_1,c_2,$c_3 \in \mathbb{N}^+$ such that $\mathbf{z}' = c_1 \mathbf{z} + c_2 \mathbf{h}$ and $\gamma' \mathbf{z}' = c_1 \gamma \mathbf{z} + c_3 \mathbf{h}$. Then we replace $\mathbf{z}'$ in the second equation

$$\begin{aligned} &\gamma'(c_1 \mathbf{z} + c_2 \mathbf{h}) = c_1 \gamma \mathbf{z} + c_3 \mathbf{h} \\ \Leftrightarrow\ &(\gamma' - \gamma) c_1 \mathbf{z} = (c_3 - \gamma' c_2) \mathbf{h} \end{aligned}$$

This however can only be true if $\gamma' - \gamma = 0$ because $\mathbf{z}$ is linearly independent of $\mathbf{h}$.

Since $\mathbf{a}$ and $\mathbf{b}$ are chosen uniformly at random then so are f,f', d_i, and d'_i. If d_i and d'_i are uniformly random then $\mathsf{Decap}(\mathsf{sk}, x, \tau) = \mathsf{Ext}(\mathbf{d}^t, r)$ and $\mathsf{Decap}(\mathsf{sk}, x', \tau') = \mathsf{Ext}(\mathbf{d'^t}, r)$ are statistically close to uniformly random by the extractor property.

Parameters. For the same language as in Construction 2 with public parameters of size $k^{1/3} \cdot \mathsf{poly}(\lambda)$ and elements of $k^{2/3} \cdot \mathsf{poly}(\lambda)$ construction 3 roughly results in public keys of size $2k^{1/3} \cdot \mathsf{poly}(\lambda)$ and an encapsulated key of size λ.

6 Incompressible PKE from Incompressible SKE and HPS

First we extend the incompressible encryption security notion [GWZ22] to the chosen ciphertext scenario and then we show a new construction paradigm using hash proof systems and incompressible symmetric-key encryption.

6.1 CCA Incompressible Encryption

We use the definition of incompressible encryption by Guan et al.[GWZ22]. It defines a public-key encryption scheme where the adversary has to know most of the ciphertext to decrypt it even with access to the secret key.

Definition 12 (Incompressible PKE)**.** *An incompressible public-key encryption scheme is a triple of PPT algorithms*

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^S)$**:** *Given the security parameter λ and a space bound S the key-generation algorithm outputs a public key pk and a secret key sk.*

$c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$**:** *Given a public key pk and a message m the encryption algorithm outputs a ciphertext c.*

$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$**:** *Given a secret key sk and a ciphertext c the decryption algorithm outputs a message m.*

Both size of message space and size of ciphertext space are polynomials over security parameter λ and space bound S, that is, $n = n(\lambda, S)$ and $l = l(\lambda, S)$ respectively.

Correctness. For all $\lambda, S \in \mathbb{N}$, messages m and $(\mathsf{pk}, \mathsf{sk})$ in the range of KeyGen we have that $\mathsf{m} = \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{m}))$.

CCA Incompressible Security. Similar to standard IND-CCA (sometimes referred to as IND-CCA2) security we extend incompressible encryption such that the adversary has access to an encryption oracle.

For security parameter λ and space bound S, a public key encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ has incompressible CCA PKE security if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ wins the following experiment with probability $\leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

$\mathsf{Dist}^{\mathsf{CCAIncomPKE}}_{\mathcal{A},\Pi}(\lambda, S)$ **Experiment** :

- Run key generation algorithm $\mathsf{KeyGen}(1^\lambda, 1^S)$ to obtain $(\mathsf{pk}, \mathsf{sk})$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk})$ on public key pk with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$ \{0, 1\}$ uniformly at random.
- Run $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_b)$ to encrypt m_b.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ for all inputs but c to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

Rate We define the rate by $\frac{|\mathsf{m}|}{|\mathsf{Enc}(\mathsf{pk},\mathsf{m})|}$ the size of a message divided by a ciphertext encrypting the message. We say a scheme has rate-1 when the rate is $1 - o(1)$.

6.2 Construction

We construct a encryption scheme that very much resembles the classic Cramer-Shoup [CS02] scheme. Instead of masking the ciphertext with the randomness that comes out of the hash proof system we use it as a key for an incompressible symmetric-key encryption scheme.

Construction 4 (Incompressible PKE)**.** *Given security parameter λ, space bound S, and message length n let (KeyGen', Encap', Decap', $\mathsf{Program}'$) be a Y-programmable hash proof system for a language $\mathcal{L} \subset X$ (where you can sample x with according witness from $\mathcal{L}$ and sample x with according trapdoor from Y) where the representation size of X is $p(\lambda, S, n)$ and encapsulated keys of size $k(\lambda, S_{\mathsf{sym}}, n)$, ($\mathsf{KeyGen}''$, Encap'', Decap'') is a 2-smooth hash proof system for the same language with encapsulation key size of λ and public key size $p'(\lambda, S, n)$, and ($\mathsf{Enc}_{\mathsf{sym}}$, $\mathsf{Dec}_{\mathsf{sym}}$) be an incompressible SKE with messages of size n, keys of size $k(\lambda, S_{\mathsf{sym}}, n)$ and ciphertexts of size $l(\lambda, S_{\mathsf{sym}}, n)$ with incompressible SKE adversary being allowed to leak a state of size $S_{\mathsf{sym}} = S + p(\lambda, S, n) + p'(\lambda, S, n)$.*

$\mathsf{KeyGen}(1^\lambda, 1^S)$**:**

- *Generate language and corresponding trapdoor* $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- *Let* $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- *Let* $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}''(\mathsf{p})$.
- *Return* $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$ *and* $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.

$\mathsf{Enc}(\mathsf{pk}, \mathsf{m})$**:**

- *Parse* $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$
- *Let* $(x, w) \leftarrow \mathsf{samp}\mathcal{L}(\mathsf{p})$.
- *Let* $\mathsf{k} \leftarrow \mathsf{Encap}'(\mathsf{pk}', x, w)$.
- *Let* $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m})$.
- *Let* $\pi \leftarrow \mathsf{Encap}''(\mathsf{pk}'', x, w, c_{\mathsf{sym}})$.
- *Return* $c = (x, c_{\mathsf{sym}}, \pi)$.

$\mathsf{Dec}(\mathsf{sk}, c)$**:**

- *Parse* $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.
- *Parse* $c = (x, c_{\mathsf{sym}}, \pi)$.
- *If* $\pi = \mathsf{Decap}''(\mathsf{sk}'', x, c_{\mathsf{sym}})$
  - *Let* $\mathsf{k} \leftarrow \mathsf{Decap}'(\mathsf{sk}', x)$
  - *Return* $\mathsf{m} = \mathsf{Dec}_{\mathsf{sym}}(\mathsf{k}, c_{\mathsf{sym}})$.
- *Return* $\perp$.

Parameters. $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is an incompressible PKE with messages of size n, ciphertexts of size $l(\lambda, S_{\mathsf{sym}}, n) + p(\lambda, S, n) + p'(\lambda, S, n)$, the adversary is allowed a leak of size $S = S_{\mathsf{sym}} - p(\lambda, S, n) - p'(\lambda, S, n)$, and the public key is of size $p(\lambda, S, n) + p'(\lambda, S, n)$.

When instantiating the two hash proof systems with constructions 2,3 and the incompressible SKE with construction 1 then $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is an incompressible PKE with messages of size n, ciphertexts of size $(n + n^{2/3}\mathsf{poly}(\lambda))(1 + o(1))$, the adversary is allowed a leak of size $S = n(1 - o(1)) - \mathsf{poly}(\lambda)(n(1 + o(1)))^{2/3}$, the public key is of size $n^{2/3}(1 + o(1))\mathsf{poly}(\lambda)$, and the secret key is of size $n(1 + o(1))\mathsf{poly}(\lambda)$.

Correctness. Follows from the correctness of (KeyGen', Encap', Decap', $\mathsf{Program}'$), (KeyGen'', Encap'', Decap''), and ($\mathsf{Enc}_{\mathsf{sym}}$, $\mathsf{Dec}_{\mathsf{sym}}$).

Theorem 4 (Security)**.** *The PKE construction 4 has incompressible CCA PKE security if (KeyGen', Encap', Decap', $\mathsf{Program}'$) is a programmable hash proof system with the listed parameters, (KeyGen'', Encap'', Decap'') is a 2-smooth hash proof system with the listed parameters, and ($\mathsf{Enc}_{\mathsf{sym}}$, $\mathsf{Dec}_{\mathsf{sym}}$) is an incompressible secure SKE with the listed parameters.*

Proof. We prove security via hybrids. First we list the hybrids and then argue their indistinguishability. In each hybrid we highlight the changes compared to the previous one.

$H_0(\lambda, S)$:

- Run key generation algorithm $\mathsf{KeyGen}(1^\lambda, 1^S)$ to obtain $(\mathsf{pk}, \mathsf{sk})$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk})$ on public key pk with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$ \{0,1\}$ uniformly at random.
- Run $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_b)$ to encrypt m_b.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ for all inputs but c to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_1 we explicitly represent what happens in KeyGen and Enc.

$H_1(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}''(\mathsf{p})$.
- Let $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$ and $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk})$ on public key pk with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$ \{0,1\}$ uniformly at random.
- Let $(x, w) \leftarrow \mathsf{samp}\mathcal{L}(\mathsf{p})$.
- Let $\mathsf{k} \leftarrow \mathsf{Encap}'(\mathsf{pk}', x, w)$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m}_b)$.
- Let $\pi \leftarrow \mathsf{Encap}''(\mathsf{pk}'', x, w, c_{\mathsf{sym}})$.
- Let $c = (x, c_{\mathsf{sym}}, \pi)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ for all inputs but c to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_2 we use the decapsulation mechanisms to encrypt the challenge message instead of encapsulation.

$H_2(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}''(\mathsf{p})$.

- Let $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$ and $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk})$ on public key pk with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$\, \{0,1\}$ uniformly at random.
- Let $(x, w) \leftarrow \mathsf{samp}\mathcal{L}(\mathsf{p})$.
- Let $\mathsf{k} \leftarrow \mathsf{Decap}'(\mathsf{sk}', x)$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m}_b)$.
- Let $\pi \leftarrow \mathsf{Decap}''(\mathsf{sk}'', x, c_{\mathsf{sym}})$.
- Let $c = (x, c_{\mathsf{sym}}, \pi)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ for all inputs but c to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_3 we sample x from $Y \subset X \setminus \mathcal{L}$ instead of $\mathcal{L}$.

$H_3(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}''(\mathsf{p})$.
- Let $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$ and $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk})$ on public key pk with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$\, \{0,1\}$ uniformly at random.
- Let $(x, \mathsf{td}_x) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$.
- Let $\mathsf{k} \leftarrow \mathsf{Decap}'(\mathsf{sk}', x)$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m}_b)$.
- Let $\pi \leftarrow \mathsf{Decap}''(\mathsf{sk}'', x, c_{\mathsf{sym}})$.
- Let $c = (x, c_{\mathsf{sym}}, \pi)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ for all inputs but c to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_4 we change the behaviour of the decryption oracle.

$H_4(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}''(\mathsf{p})$.
- Let $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$ and $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.

- Define an inefficient decryption algorithm Dec'.
  $\mathsf{Dec}'(c)$:
  - Parse $c = (x, c_{\mathsf{sym}}, \pi)$.
  - If $x \in \mathcal{L}$
    * Let w be the witness for x.
    * Let $\mathsf{k} \leftarrow \mathsf{Encap}(\mathsf{pk}', x, w)$.
    * Return $\mathsf{Dec}_{\mathsf{sym}}(\mathsf{k}, c_{\mathsf{sym}})$.
  - Else
    * Return $\perp$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}'}(\mathsf{pk})$ on public key pk with oracle access to Dec' to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$\, \{0,1\}$ uniformly at random.
- Let $(x, \mathsf{td}_x) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$.
- Let $\mathsf{k} \leftarrow \mathsf{Decap}'(\mathsf{sk}', x)$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m}_b)$.
- Let $\pi \leftarrow \mathsf{Decap}''(\mathsf{sk}'', x, c_{\mathsf{sym}})$.
- Let $c = (x, c_{\mathsf{sym}}, \pi)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}'}(\mathsf{pk}, c, \mathsf{st}_1)$ with oracle access to Dec' for all inputs but c to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_5 we program the secret key given to the adversary to decapsulate the ciphertext to the randomly chosen key k.

$H_5(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}''(\mathsf{p})$.
- Let $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$ and $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.
- Define an inefficient decryption algorthim Dec'.
  $\mathsf{Dec}'(c)$:
  - Parse $c = (x, c_{\mathsf{sym}}, \pi)$.
  - If $x \in \mathcal{L}$
    * Let w be the witness for x.
    * Let $\mathsf{k} \leftarrow \mathsf{Encap}(\mathsf{pk}', x, w)$.
    * Return $\mathsf{Dec}_{\mathsf{sym}}(\mathsf{k}, c_{\mathsf{sym}})$.
  - Else
    * Return $\perp$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}'}(\mathsf{pk})$ on public key pk with oracle access to Dec' to receive two messages m_0, m_1 and state st_1.

- Sample bit $b \leftarrow\!\!\$ \{0,1\}$ uniformly at random.
- Let $(x, \mathsf{td}_x) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$.
- Sample $\mathsf{k} \leftarrow\!\!\$ \{0,1\}^{k(\lambda, S_{\mathsf{sym}}, n)}$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m})$.
- Let $\pi \leftarrow \mathsf{Decap}''(\mathsf{sk}'', x, c_{\mathsf{sym}})$.
- Let $c = (x, c_{\mathsf{sym}}, \pi)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}'}(\mathsf{pk}, c)$ with oracle access to Dec' for all inputs but c to produce a state st_2 smaller than S.
- Let $\mathsf{sk}'_{prog} \leftarrow \mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}', x, \mathsf{k})$.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk} = (\mathsf{sk}'_{prog}, \mathsf{sk}''), \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_6 we switch the decryption oracle back to the original behaviour.

$H_6(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}''(\mathsf{p})$.
- Let $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$ and $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk})$ on public key pk with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$ \{0,1\}$ uniformly at random.
- Let $(x, \mathsf{td}_x) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_L)$.
- Sample $\mathsf{k} \leftarrow\!\!\$ \{0,1\}^{k(\lambda, S_{\mathsf{sym}}, n)}$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m})$.
- Let $\pi \leftarrow \mathsf{Decap}''(\mathsf{sk}'', x, c_{\mathsf{sym}})$.
- Let $c = (x, c_{\mathsf{sym}}, \pi)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ for all inputs but c to produce a state st smaller than S.
- Let $\mathsf{sk}'_{prog} \leftarrow \mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}', x, \mathsf{k})$.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk} = (\mathsf{sk}'_{prog}, \mathsf{sk}''), \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

$H_0 \approx H_1$:
The differences between H_0 and H_1 are purely syntactical. In H_1 we just show more detail of KeyGen and Enc.

$H_1 \approx H_2$:
In H_2 we merely change how the challenge ciphertext is calculated. By the correctness of the hash proof system these two hybrids look identical to the adversary.

$H_2 \approx_c H_3$:

In H_3 sample x from $Y \subset X \setminus \mathcal{L}$ instead of $\mathcal{L}$. These two hybrids are computationally indistinguishable by the language indistinguishability. Assume there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ that can distinguish the two hybrids H_2 and H_3 with a non-negligible advantage of ϵ. From this we construct a statistical adversary $\mathcal{A}'$ that can break language indistinguishability of the HPS with advantage ϵ.

$\mathcal{A}'(\mathsf{p}, x)$:

- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}''(\mathsf{p})$.
- Let $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$ and $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk})$ on public key pk with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$ \{0, 1\}$ uniformly at random.
- Let $\mathsf{k} \leftarrow \mathsf{Decap}'(\mathsf{sk}', x)$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m}_b)$.
- Let $\pi \leftarrow \mathsf{Decap}''(\mathsf{sk}'', x, c_{\mathsf{sym}})$.
- Let $c = (x, c_{\mathsf{sym}}, \pi)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ for all inputs but c to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

If $\mathcal{A}$ can distinguish H_2 from H_3 with advantage ε then the advantage of $\mathcal{A}'$ of distinguishing a x sampled from $\mathcal{L}$ and sampling x from $Y \subset X \setminus \mathcal{L}$ is also ϵ as it perfectly simulates H_2 in the case that $x \in \mathcal{L}$ and perfectly simulates H_3 in the other case.

$H_3 \approx_s H_4$:

According to 2-smoothness of $(\mathsf{KeyGen}'', \mathsf{Encap}'', \mathsf{Decap}'')$ in the decryption oracle for $c = (x, c_{\mathsf{sym}}, \pi)$ if $x \notin \mathcal{L}$ the decryption oracle fails with $2^{-\lambda}$ probability even when the adversary has access to the challenge ciphertext $(x^*, c^*_{\mathsf{sym}}, \pi^*)$ where $x^* \in X \setminus \mathcal{L}$. Simply not answering the query if $x \in X \setminus \mathcal{L}$ is statistically close to answering with probability $2^{-\lambda}$.

$H_4 \approx_s H_5$:

According to programmable smoothness of $(\mathsf{KeyGen}', \mathsf{Encap}', \mathsf{Decap}')$ if $x \notin \mathcal{L}$ then $(\mathsf{pk}', \mathsf{sk}', x)$ is statistically close to $(\mathsf{pk}', \mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}', x, \mathsf{k}), x)$ for uniformly random k. Because this is exactly what we switch we get that H_4 and H_5 are statistically close.

$H_5 \approx_c H_6$:

Again, According to 2-smoothness of $(\mathsf{KeyGen}'', \mathsf{Encap}'', \mathsf{Decap}'')$ in the decryption oracle for $c = (x, c_{\mathsf{sym}}, \pi)$ if $x \notin \mathcal{L}$ the decryption oracle fails with $2^{-\lambda}$ probability even when the adversary has access to the challenge ciphertext $(x^*, c^*_{\mathsf{sym}}, \pi^*)$ where $x^* \in X \setminus \mathcal{L}$. Simply not answering the query if $x \in X \setminus \mathcal{L}$ is statistically close to answering with probability $2^{-\lambda}$.

$H_6 \approx \mathsf{Dist}^{\mathsf{IncomSKE}}_{\mathcal{A}', \Pi_{\mathsf{sym}}}$:

Finally, given an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ that wins experiment $H_6(\lambda, S)$ with probability ϵ we construct an adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2, \mathcal{A}'_3)$ that wins the $\mathsf{Dist}^{\mathsf{IncomSKE}}_{\mathcal{A}', \Pi_{\mathsf{sym}}}(\lambda, S + p(\lambda) + p'(\lambda))$ experiment with probability ϵ.

$\mathcal{A}'_1(1^\lambda, 1^S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_L) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.

- Let $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}''(\mathsf{p})$.
- Let $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$ and $\mathsf{sk} = (\mathsf{sk}', \mathsf{sk}'')$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk})$ on public key pk with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ to receive two messages m_0, m_1 and state st_1.
- Let $\mathsf{st}_1' \leftarrow (\mathsf{pk}, \mathsf{sk}, \mathsf{td}_{\mathcal{L}}, \mathsf{st}_1)$.
- Return m_0, m_1, and st_1'

$\mathcal{A}_2'(\mathsf{st}_1', c_{\mathsf{sym}})$:

- Parse $\mathsf{st}_1' = (\mathsf{pk}, \mathsf{sk}, \mathsf{td}_{\mathcal{L}}, \mathsf{st}_1)$
- Let $(x, \mathsf{td}_x) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_L)$
- Let $\pi \leftarrow \mathsf{Decap}''(\mathsf{sk}'', x)$
- Let $c \leftarrow (x, c_{\mathsf{sym}}, \pi)$
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ with oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ for all inputs but c to produce a state st_2 smaller than S
- Return state $\mathsf{st}_2' \leftarrow (x, \mathsf{td}_x, \mathsf{st}_2)$ smaller than $S_{\mathsf{sym}} = S + p(\lambda) + p'(\lambda)$

$\mathcal{A}_3'(\mathsf{k}, \mathsf{st}_1', \mathsf{st}_2', \mathsf{m}_0, \mathsf{m}_1)$:

- Parse $\mathsf{st}_1 = (\mathsf{pk}, \mathsf{sk} = (\mathsf{sk}', \mathsf{sk}''), \mathsf{td}_{\mathcal{L}}, \mathsf{st}_1')$
- Parse $\mathsf{st}_2 = (x, \mathsf{td}_x, \mathsf{st}_2')$
- Program $\mathsf{sk}'_{prog} \leftarrow \mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}', x, \mathsf{k})$
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{pk}, (\mathsf{sk}'_{prog}, \mathsf{sk}''), \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$
- Return b'

$\mathcal{A}'$ wins $\mathsf{Dist}^{\mathsf{IncomSKE}}_{\mathcal{A}', \Pi_{\mathsf{sym}}}(\lambda, S + p(\lambda))$ iff $\mathcal{A}$ wins in $H_6(\lambda, S)$ because $\mathcal{A}'$ perfectly simulates H_6 from the perspective of $\mathcal{A}$.

□

7 The Dangers of Using the ROM in Incompressible Encryption

In this section, we show that there is a very simple incompressible encryption scheme that is secure in the ROM. However, as soon as we instantiate the random oracle with a specific hash function, the scheme is not secure anymore. Through this we demonstrate a proof in the ROM might be meaningless and extra precautions must be taken before designing schemes in the ROM. This provides a fairly natural example on the uninstantiability of a random oracle. The technique has similarities to observations by [DM04] about inital key generation for the bounded storage model.

7.1 Construction

The construction we present in this work is secure in the ideal cipher model. The ideal cipher model provides a oracle to a keyed function P such that for every key k we have that P_{k} is an independent random permutation. It is proven to be equivalent to the random oracle model [HKT11].

Construction 5. *Let* $(\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ *be an IND-CPA encryption scheme with a secret key size* $p(\lambda)$ *and* $\mathsf{P}_{\mathsf{k}} : \{0,1\}^{\mu} \to \{0,1\}^{\mu}$ *be a random permutation indexed by* k *modeled as an ideal cipher, where* $\mu = p(\lambda) + \lambda + S$.

$\mathsf{KeyGen}(1^{\lambda})$**:**

- *Compute* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}'(1^{\lambda})$.
- *Return* pk *and* sk.

$\mathsf{Enc}(\mathsf{pk}, \mathsf{m} \in \{0,1\}^S)$**:**

- *Sample* $\mathsf{k} \leftarrow\!\!\$\, \{0,1\}^\lambda$
- *Encrypt* $c' \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathsf{k})$.
- *Sample* $r \leftarrow\!\!\$\, \{0,1\}^\lambda$ *uniformly at random*
- *Let* $d \leftarrow \mathsf{P}_\mathsf{k}((r, \mathsf{m}))$.
- *Return* $c = (c', d)$.

$\mathsf{Dec}(\mathsf{sk}, c)$**:**

- *Parse* $c = (c', d)$
- *Decrypt* $\mathsf{k} \leftarrow \mathsf{Dec}'(\mathsf{sk}', c')$.
- *Compute* $(r, \mathsf{m}) = \mathsf{P}_\mathsf{k}^{-1}(d)$.
- *Output* m.

Correctness Correctness follows trivially from the correctness of the underlying PKE and by the fact that $\mathsf{P}_\mathsf{k}^{-1}(\mathsf{P}_\mathsf{k}(r, \mathsf{m})) = (r, \mathsf{m})$. We now analyze the security of the scheme.

Theorem 5. *The scheme presented in Construction 5 has incompressible PKE security in the random oracle model.*

Proof. We prove the theorem via the following hybrids.

H_0 : This is the incompressible PKE security game.

- Run key generation algorithm $\mathsf{KeyGen}'(1^\lambda)$ to obtain $(\mathsf{pk}, \mathsf{sk})$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ to receive two messages m_0 and m_1 with oracle access to P and P^{-1}.
- Sample bit $b \leftarrow\!\!\$\, \{0,1\}$ uniformly at random.
- Sample $\mathsf{k} \leftarrow\!\!\$\, \{0,1\}^\lambda$.
- Encrypt $c' \leftarrow \mathsf{Enc}'(\mathsf{pk}, \mathsf{k})$.
- Sample $r \leftarrow\!\!\$\, \{0,1\}^\lambda$ uniformly at random.
- Compute $d \leftarrow \mathsf{P}_\mathsf{k}(r, \mathsf{m}_b)$.
- Let $c = (c', d)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S with oracle access to P and P^{-1}.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$ with oracle access to P and P^{-1}.
- The adversary wins if $b = b'$.

In this hybrid, the experiment aborts if P_k is ever queried by the first stage adversary.

H_1 :

- Run key generation algorithm $\mathsf{KeyGen}'(1^\lambda)$ to obtain $(\mathsf{pk}, \mathsf{sk})$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ to receive two messages m_0 and m_1 with oracle access to P and P^{-1}.

- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random.
- Sample $\mathsf{k} \leftarrow\$ \{0,1\}^\lambda$.
- Encrypt $c' \leftarrow \mathsf{Enc}'(\mathsf{pk}, \mathsf{k})$.
- Sample $r \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Compute $d \leftarrow \mathsf{P}_\mathsf{k}(r, \mathsf{m}_b)$.
- Let $c = (c', d)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S with oracle access to P and P^{-1}.
- If P_k is queried by $\mathcal{A}_1$ or $\mathcal{A}_2$, abort.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$ with oracle access to P and P^{-1}.
- The adversary wins if $b = b'$.

In this hybrid, the experiment aborts if the final stage adversary ever queries $\mathsf{P}_\mathsf{k}^{-1}$ on the value d.

H_2 :

- Run key generation algorithm $\mathsf{KeyGen}'(1^\lambda)$ to obtain $(\mathsf{pk}, \mathsf{sk})$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ to receive two messages m_0 and m_1 with oracle access to P and P^{-1}.
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random.
- Sample $\mathsf{k} \leftarrow\$ \{0,1\}^\lambda$.
- Encrypt $c' \leftarrow \mathsf{Enc}'(\mathsf{pk}, \mathsf{k})$.
- Sample $r \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.
- Compute $d \leftarrow \mathsf{P}_\mathsf{k}(r, \mathsf{m}_b)$.
- Let $c = (c', d)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S with oracle access to P and P^{-1}.
- If P_k is queried by $\mathcal{A}_1$ or $\mathcal{A}_2$ abort.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$ with oracle access to P and P^{-1}.
- If $\mathcal{A}_3$ queries $\mathsf{P}_\mathsf{k}^{-1}(d)$ before ever querying $\mathsf{P}_\mathsf{k}((r, \mathsf{m}_b))$ abort.
- The adversary wins if $b = b'$.

In this hybrid, the experiment aborts if the adversary queried P_k on (r, m_b). This removes almost all information about m_b.

H_3 :

- Run key generation algorithm $\mathsf{KeyGen}'(1^\lambda)$ to obtain $(\mathsf{pk}, \mathsf{sk})$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ to receive two messages m_0 and m_1 with oracle access to P and P^{-1}.
- Sample bit $b \leftarrow\$ \{0,1\}$ uniformly at random.
- Sample $\mathsf{k} \leftarrow\$ \{0,1\}^\lambda$.
- Encrypt $c' \leftarrow \mathsf{Enc}'(\mathsf{pk}, \mathsf{k})$.
- Sample $r \leftarrow\$ \{0,1\}^\lambda$ uniformly at random.

- Compute $d \leftarrow \mathsf{P}_\mathsf{k}(r, \mathsf{m}_b)$.
- Let $c = (c', d)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ to produce a state st_2 smaller than S with oracle access to P and P^{-1}.
- If P_k is queried by $\mathcal{A}_1$ or $\mathcal{A}_2$, abort.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$ with oracle access to P and P^{-1}.
- If $\mathcal{A}_3$ queries $\mathsf{P}_\mathsf{k}^{-1}(d)$ before ever querying $\mathsf{P}_\mathsf{k}((r, \mathsf{m}_b))$ abort.
- If $\mathcal{A}_3$ queries $\mathsf{P}_\mathsf{k}((r, \mathsf{m}_b))$ abort.
- The adversary wins if $b = b'$.

We now show indistinguishability of hybrids.

$H_0 \approx_c H_1$:

The hybrids are identical except for the abort condition. The experiment aborts the protocol if the ideal cipher P_k is ever queried by $\mathcal{A}_1$ or $\mathcal{A}_2$. We argue that the winning probability in H_0 and H_1 are negligibly close if (KeyGen', Enc', Dec') is an IND-CPA secure PKE.

Assume there is an adversary $\mathcal{A}_1$, $\mathcal{A}_2$ that queries k with probability ε. From this, we construct an adversary $\mathcal{A}'_1$, $\mathcal{A}'_2$ that breaks the IND-CPA security of the encryption scheme (KeyGen', Enc', Dec').

$\mathcal{A}'_1(\mathsf{pk})$:

- Sample $\mathsf{k}_0, \mathsf{k}_1 \leftarrow\!\!\$\, \{0,1\}^\lambda$.
- Also call $\mathsf{k}_0 = \mathsf{k}$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ to receive two messages m_0 and m_1 with oracle access to P and P^{-1}.
- Return k_0 and k_1.

$\mathcal{A}'_2(\mathsf{pk}, c')$:

- Sample $r \leftarrow\!\!\$\, \{0,1\}^\lambda$ uniformly at random.
- Sample $b' \leftarrow\!\!\$\, \{0,1\}$ uniformly at random.
- Let $d \leftarrow \mathsf{P}_\mathsf{k}((r, \mathsf{m}_{b'}))$.
- Let $c = (c', d)$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{st}_1, c)$ with oracle access to P and P^{-1}.
- If $\mathcal{A}_1$ or $\mathcal{A}_2$ ever queried P_k return 0 else 1.

If $b = 0$ then ($\mathcal{A}'_1$, $\mathcal{A}'_2$) outputs 0 with probability ε as this perfectly simulates H_0 to the adversary ($\mathcal{A}_1$, $\mathcal{A}_2$).

If $b = 1$ then ($\mathcal{A}'_1$, $\mathcal{A}'_2$) will output 1 with probability $1 - \mathsf{negl}(\lambda)$ as c contains no information about k beyond $d = \mathsf{P}_\mathsf{k}((r, \mathsf{m}_{b'}))$.

Combining the two cases the probability of winning $\mathcal{A}'$ winning the IND-CPA experiment is $\frac{1}{2}\varepsilon + \frac{1}{2}(1 - \mathsf{negl}(\lambda))$. Which is a non-negligible advantage if ε is non-negligible.

$H_1 \approx H_2$:

The hybrids are identical except for the new abort condition. We argue that the winning probability in H_1 and H_2 are negligibly close if the adversary can only polynomially many queries $q(\lambda)$.

Let D, SK, ST_2 be the random variables that reflect the occurrence of d, sk, st_2 in the execution of the experiment.

We know that $H_\infty(D) = S + \lambda + p(\lambda)$ because without access to the permutation P_k (which $\mathcal{A}_1$ and $\mathcal{A}_2$ don't have) we have d is a uniformly random string.

We are going to show that $\tilde{H}_\infty(D|SK, ST_2) \geq \lambda$. We have that

$$\begin{aligned}\tilde{H}_\infty(D|SK, ST_2) &\geq H_\infty(D) - p(\lambda) - S \\ &\geq S + \lambda + p(\lambda) - p(\lambda) - S = \lambda\end{aligned}$$

where the first inequality follows from Lemma 1 and the last step follows from the fact that D is a uniform vector over $\mu = S + \lambda + p(\lambda)$.

Hence the adversary with $q(\lambda)$ queries can at most query $\mathsf{P}_\mathsf{k}^{-1}$ on d with probability $q(\lambda)/2^\lambda$. Therefore, the winning probability of H_1 only be better than that of H_2 by $q(\lambda)/2^\lambda$.

$H_2 \approx H_3$:

The only difference between H_2 and H_3 is additional abort condition if $\mathcal{A}_3$ queries $\mathsf{P}_k((r, \mathsf{m}_b))$. We argue that the winning probability in H_2 and H_3 are negligibly close if the adversary can only polynomially many queries $q(\lambda)$.

The abort condition, however, can only be reached if previous abort conditions are not met. So the adversary needs to guess r, which is uniformly random in $\{0,1\}^\lambda$. Therefore, if the adversary has $q(\lambda)$ queries to the random oracle, she has at most reached the new abort condition of querying $\mathsf{P}_k((r, \mathsf{m}_b))$ with probability of $q(\lambda)/2^\lambda$.

H_3 :

In H_3 the adversary learns no information about m_b because it can never query $\mathsf{P}_k((r, \mathsf{m}_b))$ or $\mathsf{P}_k^{-1}(d)$ and all other queries are uniformly random subject to being different from all other queries and $\mathsf{P}_k((r, \mathsf{m}_b)) = d$.

□

7.2 Attack

We instantiate the ideal cipher with a specific keyed permutation P. Moreover, the only condition we have on the underlying PKE is that it is IND-CPA secure. Hence, we can instantiate it with an FHE scheme $(\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Eval}', \mathsf{Dec}')$. The idea of the attack is that, as soon as the ideal cipher is a specific permutation P_k, the adversary can make non-black-box use of the permutation. It homomorphically evaluates the permutation inside the FHE allowing it to compress the ciphertext into an encryption of a single bit.

We now provide the description of the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$.

$\mathcal{A}_1(\mathsf{pk})$: Output any distincit two messages $\mathsf{m}_0, \mathsf{m}_1$.

$\mathcal{A}_2(c)$:

- Parse $c = (c', d)$.
- Consider the circuit $\mathcal{C}(x) = \mathcal{C}(x)_{\mathsf{P},d,\mathsf{m}_0}$ which does the following:
  1. Evaluate $w = \mathsf{P}_\mathsf{k}^{-1}(d)$.
  2. If $w = \mathsf{m}_0$, output 0. Otherwise, output 1.
- Compute $\tilde{c} \leftarrow \mathsf{Eval}'(\mathsf{pk}, \mathcal{C}, c')$.
- Output $\mathsf{st} = \tilde{c}$.

$\mathcal{A}_3(\mathsf{st}, \mathsf{sk})$: Compute $b' \leftarrow \mathsf{Dec}'(\mathsf{sk}, \tilde{c})$. Output b'.

We can establish that $b' = b$ by the homomorphic correctness of the FHE.

Size of st. The state st of the adversary is composed by a single FHE ciphertext encrypting one bit. Hence, by the compactness of the FHE $|\mathsf{st}| = \mathsf{poly}(\lambda)$.

Remark 1. *Notice encrypting the key of any (plain model) incompressible encryption scheme in an FHE and adding this to the public key removes the incompressibility by the same argument. This observation demonstrates the dangers of using the ROM in incompressible encryption and that incompressible encryption is a fairly delicate notion.*

References

[ABP15] Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Disjunctions for hash proof systems: New constructions and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 69–100, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[ADMP20] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 411–439, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

[ADN+10] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 113–134, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 36–54, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[AR99] Yonatan Aumann and Michael O. Rabin. Information theoretically secure communication in the limited storage space model. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 65–79, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[BD17] Mihir Bellare and Wei Dai. Defending against key exfiltration: Efficiency improvements for big-key cryptography via large-alphabet subkey prediction. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 923–940, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[BFM15] Christina Brzuska, Pooya Farshim, and Arno Mittelbach. Random-oracle uninstantiability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 428–455, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BKR16] Mihir Bellare, Daniel Kane, and Phillip Rogaway. Big-key symmetric encryption: Resisting key exfiltration. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 373–402, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[Bla06] John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In Matthew J. B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 328–340, Graz, Austria, March 15–17, 2006. Springer, Heidelberg, Germany.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[CFPZ09] Céline Chevalier, Pierre-Alain Fouque, David Pointcheval, and Sébastien Zimmer. Optimal randomness extraction from a Diffie-Hellman element. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 572–589, Cologne, Germany, April 26–30, 2009. Springer, Heidelberg, Germany.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, jul 2004.

[CM97] Christian Cachin and Ueli M. Maurer. Unconditional security against memory-bounded adversaries. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 292–306, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 45–64, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2003.

[Den02] Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 100–109, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany.

[DGO19] Ivan Damgård, Chaya Ganesh, and Claudio Orlandi. Proofs of replicated storage without timing assumptions. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 355–380, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer, Heidelberg, Germany.

[DLW06] Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 225–244, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.

[DM04] Stefan Dziembowski and Ueli M. Maurer. On generating the initial key in the bounded-storage model. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 126–137, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.

[DQW21] Yevgeniy Dodis, Willy Quach, and Daniel Wichs. Speak much, remember little: Cryptography in the bounded storage model, revisited. Cryptology ePrint Archive, Report 2021/1270, 2021. `https://eprint.iacr.org/2021/1270`.

[Dzi06a] Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 207–224, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.

[Dzi06b] Stefan Dziembowski. On forward-secure storage (extended abstract). In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 251–270, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th Annual Symposium on Foundations of Computer Science*, pages 102–115, Cambridge, MA, USA, October 11–14, 2003. IEEE Computer Society Press.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th Annual Symposium on Foundations of Computer Science*, pages 612–621, Berkeley, CA, USA, October 15–17, 2017. IEEE Computer Society Press.

[GLW20] Rachit Garg, George Lu, and Brent Waters. New techniques in replica encodings with client setup. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part III*, volume 12552 of *Lecture Notes in Computer Science*, pages 550–583, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 162–179, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[GWZ22] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Incompressible cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 700–730, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany.

[GZ19] Jiaxin Guan and Mark Zhandry. Simple schemes in the bounded storage model. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 500–524, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[GZ21] Jiaxin Guan and Mark Zhandry. Disappearing cryptography in the bounded storage model. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 365–396, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HKT11] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 89–98, San Jose, CA, USA, June 6–8, 2011. ACM Press.

[HLWW13] Carmit Hazay, Adriana López-Alt, Hoeteck Wee, and Daniel Wichs. Leakage-resilient cryptography from minimal assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 160–176, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, pages 60–73. ACM, 2021.

[Kal05] Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 78–95, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

[LMQW22] Alex Lombardi, Ethan Mook, Willy Quach, and Daniel Wichs. Post-quantum insecurity from lwe. *Cryptology ePrint Archive*, 2022.

[Mau92] Ueli Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Jornal of Cryptology*, 5(1):53–66, Jan 1992.

[Mau93] Ueli M. Maurer. Protocols for secret key agreement by public discussion based on common information. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 461–470, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.

[MW20] Tal Moran and Daniel Wichs. Incompressible encodings. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 494–523, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[Raz17] Ran Raz. A time-space lower bound for a large class of learning problems. In Chris Umans, editor, *58th Annual Symposium on Foundations of Computer Science*, pages 732–742, Berkeley, CA, USA, October 15–17, 2017. IEEE Computer Society Press.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

A Programmable Hash Proof System from wPR-EGA

We show that the hash proof system based on weak pseudorandom effective group actions of [ADMP20] is programmable. Weak pseudorandom effective group actions can be instantiated from isogeny based assumptions.

A.1 Cryptographic Group Actions

We quickly recap what group actions are to understand our notation. We had to slightly modify the notation of [ADMP20] to match our group notation.

Definition 13 (Group Action)**.** *A group $\mathbb{G}$ is said to act on a set $\mathcal{X}$ if there is a map $\star : \mathbb{G} \times \mathcal{X} \to \mathcal{X}$ that satisfies the following two properties:*

Identity *: If $[0]$ is the identity element of $\mathbb{G}$, then for any $x \in \mathcal{X}$, we have $[0] \star x = x$*

Compatibility *: For any $[g], [h] \in \mathbb{G}$ and any $x \in \mathcal{X}$, we have $([g] + [h]) \star x = [g] \star ([h] \star x)$*

In the following construction the require the group action to be regular and to be a weakly pseudorandom effective group action. The details about these properties can be found in [ADMP20].

A.2 Construction

Construction 6. *Using weak pseudorandom effective group actions and a randomness extractor $\mathsf{Ext} : \mathcal{X}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ we get a programmable hash proof system with an encapsulated key size of 1 bit.*

$\mathsf{Gen}(1^\lambda)$ *:*

- *Sample $\overline{x}_0 \leftarrow\!\!\$\, \mathcal{X}$ uniformly at random*
- *Sample $[z] \leftarrow\!\!\$\, \mathbb{G}$ uniformly at random*
- *Let $\overline{x}_1 \leftarrow [z] \star \overline{x}_0$*

- *Return* $\mathsf{p} = (\overline{x}_0, \overline{x}_1)$ *and* $\mathsf{td}_{\mathcal{L}} = [z]$

$\mathsf{samp}\mathcal{L}(\mathsf{p})$ *:*

- *Parse* $\mathsf{p} = (\overline{x}_0, \overline{x}_1)$
- *Sample* $[g] \leftarrow\!\!\$\, \mathbb{G}$ *uniformly at random*
- *Let* $(x_0, x_1) \leftarrow ([g] \star \overline{x}_0, [g] \star \overline{x}_1)$
- *Return* $x = (x_0, x_1)$ *and witness* $w = ([g])$

$\mathsf{samp}Y(\mathsf{p})$ *:*

- *Parse* $\mathsf{p} = (\overline{x}_0, \overline{x}_1)$
- *Sample* $[g_0] \leftarrow\!\!\$\, \mathbb{G}$ *uniformly at random*
- *Sample* $[g_1] \leftarrow\!\!\$\, \mathbb{G} \setminus \{[g_0]\}$ *uniformly at random*
- *Let* $(x_0, x_1) \leftarrow ([g_0] \star \overline{x}_0, [g_1] \star \overline{x}_1)$
- *Return* $x = (x_0, x_1)$ *and trapdoor* $\mathsf{td}_x = ([g_0], [g_1])$

$\mathsf{KeyGen}(\mathsf{p})$ *:*

- *Sample* $[\mathbf{h}] \leftarrow\!\!\$\, \mathbb{G}^{\lambda}$ *uniformly at random*
- *Sample* $\mathbf{b} \leftarrow\!\!\$\, \{0,1\}^{\lambda}$ *uniformly at random*
- *Sample* $\mathsf{s} \leftarrow\!\!\$\, \{0,1\}^{\lambda}$
- *Let* $\mathsf{sk} \leftarrow ([\mathbf{h}], \mathbf{b})$
- *Let* $\mathsf{pk} \leftarrow ([h_1] \star \overline{x}_{b_1}, \ldots, [h_\lambda] \star \overline{x}_{b_\lambda}, \mathsf{s})$
- *Return* pk *and* sk

$\mathsf{Encap}\ \ (\mathsf{pk}, x = ([x_0], [x_1]), w = [g])$*:*

- *Parse* $\mathsf{pk} = (y_1, \ldots, y_l)$
- *Let* $\mathsf{k} \leftarrow \mathsf{Ext}(([g] \star y_1, \ldots, [g] \star y_\lambda), \mathsf{s})$
- *Return* k

$\mathsf{Decap}\ \ (\mathsf{sk}, x = ([x_0], [x_1]))$*:*

- *Parse* $\mathsf{sk} = ([\mathbf{h}], \mathbf{b})$
- *Let* $\mathsf{k} \leftarrow \mathsf{Ext}(([h_1] \star x_{b_1}, \ldots, [h_\lambda] \star x_{b_\lambda}), \mathsf{s})$
- *Return* k*.*

$\mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}, x, \mathsf{k})$ *:*

- *Parse* $\mathsf{td}_{\mathcal{L}} = [z]$*,* $\mathsf{td}_x = ([g_0], [g_1])$*, and* $\mathsf{sk} = ([\mathbf{h}], \mathbf{b})$
- *Repeat*
  - *Sample* $\mathbf{b}' \leftarrow\!\!\$\, \{0,1\}^{\lambda}$ *uniformly at random*
  - *Let* $([h'_1], \ldots, [h'_\lambda]) \leftarrow ((b_1 \oplus b'_1)[z] + [h_1], \ldots, (b_\lambda \oplus b'_\lambda)[z] + [h_\lambda])$
  - *Let* $\mathsf{sk}' \leftarrow (\mathbf{b}', [\mathbf{h}'])$
- *Until* $\mathsf{k} = \mathsf{Decap}(\mathsf{sk}', x)$
- *Return* sk'

Correctness and Language Indistinguishability Correctness and language indistinguishability remain exactly the same as in [ADMP20].

Programmability and Programmable Smoothness Because the programming is just rejection sampling a secret key in exactly the same way as the original sampling but under the condition that $\mathsf{Decap}(\mathsf{sk}', x) = \mathsf{k}$ this follows from correctness and smoothness of the original scheme. Rejection sampling is efficient because the key k only has size 1.

Beyond one bit To encapsulate a key of size k one can simply generate k public-key-secret-key pairs and then encapsulation, decapsulation, and programming is done bitwise. This modification makes public and secret keys k times as big while leaving the language elements unchanged.

B Programmable Hash Proof System from LWE

We present a programmable hash proof system using lattice the trapdoors by [MP12] and rounding.

B.1 Preliminaries

Gaussian Distribution For any $\sigma \in \mathbb{R}_{>0}$ let $\rho_\sigma(\mathbf{x}) = exp(-\pi||\mathbf{x}||^2/\sigma^2)$ be the Gaussian function on $\mathbb{R}^m$ with deviation σ. Let χ_σ^m be the discrete Gaussian distribution over $\mathbb{Z}^m$ with deviation σ.

Definition 14 (LWE)**.** *The LWE assumption states that for* $\mathbf{A} \leftarrow\!\$\, \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \leftarrow\!\$\, \mathbb{Z}_q^m$, *and* $\mathbf{b} \leftarrow\!\$\, \mathbb{Z}_q^n$ *being uniformly random and* $\mathbf{e} \leftarrow \chi_\sigma^n$ *being sampled from a small gaussian distribution. We have that*

$$(\mathbf{A}, \mathbf{As} + \mathbf{e}) \approx_c (\mathbf{A}, \mathbf{b})$$

Definition 15 (Lattice Trapdoor)**.** *A lattice trapdoor consists of two PPT algorithms*

- $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$ *on inputs* $n, m, q \in \mathbb{N}$ *outputs matrices* $\mathbf{A} \in \mathbb{Z}_q^{n\times m}, \mathbf{T} \in \mathbb{Z}_q^{m\times m}$
- $\mathbf{r} \leftarrow \mathsf{SampleD}(\mathbf{A}, \mathbf{T}, \mathbf{u}, s)$ *on inputs* $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{T} \in \mathbb{Z}_q^{m\times m}$, $\mathbf{u} \in \mathbb{Z}_q^n$, $s \in$ *and outputs* $\mathbf{r} \in \mathbb{Z}_q^m$

For any $n \geq 1$, $q \geq 2$, *sufficiently large* $m = \Omega(n \log(q))$, *and sufficiently large* $\sigma = \Omega(\sqrt{n}\log(q))$ *these two distributions producing* $(\mathbf{A}, \mathbf{T}, \mathbf{u}, \mathbf{r})$ *are statistically close.*

- $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$; $\mathbf{u} \leftarrow \mathbb{Z}_q^n$; $\mathbf{r} \leftarrow \mathsf{SampleD}(\mathbf{A}, \mathbf{T}, \mathbf{u}, \sigma)$.
- $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$; $\mathbf{r} \leftarrow \chi_\sigma^m$; $\mathbf{u} \leftarrow \mathbf{Ar}$.

Also, $\mathbf{A}$ *is statistically close to uniformly random.*

B.2 Construction

Construction 7. *We construct a programmable hash proof system that is secure assuming LWE with superpolynomial modulus-to-noise ratio.*

$\mathsf{Gen}(1^\lambda)$:

- *Sample matrix* $\begin{pmatrix}\mathbf{A}\\ \mathbf{B}\end{pmatrix} \leftarrow \mathsf{TrapSamp}(1^{2n}, 1^m, q)$ *with lattice trapdoor* $\mathbf{T}$ *and* $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n\times m}$.
- *Return* $\mathsf{p} = \mathbf{A}$ *and* $\mathsf{td}_{\mathcal{L}} = (\mathbf{B}, \mathbf{T})$.

$\mathsf{samp}\mathcal{L}(\mathsf{p})$:

- *Parse* $\mathsf{p} = \mathbf{A}$.

- *Sample* $\mathbf{S} \leftarrow\!\!\$ \{0,1\}^{v \times n}$ *uniformly from the binary matrices.*
- *Sample* $\mathbf{E} \leftarrow \chi^{v \times m}$ *with small gaussian entries.*
- *Return* $x = \mathbf{SA} + \mathbf{E}$ *and* $w = (\mathbf{S}, \mathbf{E})$.

$\mathsf{samp}Y(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$ *:*

- *Parse* $\mathsf{td}_{\mathcal{L}} = (\mathbf{B}, \mathbf{T})$.
- *Return* $x = \mathbf{B}$ *and* $\mathsf{td}_x = ()$.

$\mathsf{KeyGen}(\mathsf{p})$ *:*

- *Parse* $\mathsf{p} = \mathbf{A}$.
- *Sample* $\mathbf{R} \leftarrow\!\!\$ \chi_\sigma^{m \times u}$ *with small gaussian entries.*
- *Return* $\mathsf{pk} = \mathbf{AR}$ *and* $\mathsf{sk} = \mathbf{R}$

$\mathsf{Encap}(\mathsf{pk}, x, w)$ *:*

- *Parse* $\mathsf{pk} = \mathbf{AR}$ *and* $w = (\mathbf{S}, \mathbf{E})$.
- *Let* $\mathsf{k} \leftarrow \lceil \mathbf{SAR} \rfloor$.
- *Return* k.

$\mathsf{Decap}(\mathsf{sk}, x)$ *:*

- *Parse* $\mathsf{sk} = \mathbf{R}$ *and* $x = \mathbf{SA} + \mathbf{E}$.
- *Let* $\mathsf{k} \leftarrow \lceil (\mathbf{SA} + \mathbf{E})\mathbf{R} \rfloor$.
- *Return* k.

$\mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}, x, \mathsf{k})$ *:*

- *Parse* $\mathsf{sk} = \mathbf{R}$.
- *Sample* $\mathbf{K} \in \mathbb{Z}_q^{v \times u}$ *uniformly at random conditioned on* $\lceil \mathbf{K} \rfloor = \mathsf{k}$.
- *Sample short* $\mathbf{R}^* \leftarrow \mathsf{SampleD}(\begin{pmatrix} \mathbf{A} \\ \mathbf{B} \end{pmatrix}, \mathbf{T}, \begin{pmatrix} \mathbf{AR} \\ \mathbf{K} \end{pmatrix}, \sigma)$
- *Return* $\mathsf{sk}' = \mathbf{R}^*$.

Statistical Correctness With $\mathbf{A}$,$\mathbf{S}$,$\mathbf{E}$ and $\mathbf{R}$ as in the construction, we have that $\mathsf{Encap}(\mathsf{pk}, x, w) = \lceil \mathbf{SAR} \rfloor$ which is statistically close to $\lceil (\mathbf{SA} + \mathbf{E})\mathbf{R} \rfloor = \lceil \mathbf{SAR} + \mathbf{ER} \rfloor = \mathsf{Decap}(\mathsf{sk}, x)$ when using a superpolynomial modulus-to-noise ratio.

Language Indistinguishability By LWE assumption $\mathbf{SA}+\mathbf{E}$ is indistinguishable from uniformly random. The trapdoored matrix $\mathbf{B}$ is statistically close to uniform. Therefore, language indistinguishability follows from LWE.

Programmability Using the correctness of the lattice trapdoor we have that $\mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}, x, \mathsf{k})$ returns a key sk' s.t. $\lceil \mathbf{BR}^* \rfloor = \mathsf{k}$.

Programmable Smoothness $\mathsf{sk} = \mathbf{R}$ are sampled from $\chi_\sigma^{m \times u}$ with small gaussian entries and $\mathbf{R}^*$ is statistically close to being sampled from $\chi_\sigma^{m \times u}$ conditioned on $\lceil \mathbf{BR}^* \rfloor = \mathsf{k}$ where k is uniformly random. Therefore, sk' is statistically close to sk.

B.3 Incompressible Encryption

Since we do not have a 2-smooth hash proof system for the same language as the above LWE programmable HPS, we do not achieve CCA incompressible PKE but only CPA incompressible PKE. The transfromation stays almost the same as in the CCA case.

Below we give the transfromation for CPA incompressible PKE.

Construction 8 (Incompressible PKE)**.** *Given security parameter λ, space bound S, and message length n let (KeyGen', Encap', Decap', $\mathsf{Program}'$) be a Y-programmable hash proof system for a language $\mathcal{L} \subset X$ (where you can sample x with according witness from $\mathcal{L}$ and sample x with according trapdoor from Y) where the representation size of X is $p(\lambda, S, n)$ and encapsulated keys of size $k(\lambda, S_{\mathsf{sym}}, n)$, and ($\mathsf{Enc}_{\mathsf{sym}}$, $\mathsf{Dec}_{\mathsf{sym}}$) be an incompressible SKE with messages of size n, keys of size $k(\lambda, S_{\mathsf{sym}}, n)$ and ciphertexts of size $l(\lambda, S_{\mathsf{sym}}, n)$ with incompressible SKE adversary being allowed to leak a state of size $S_{\mathsf{sym}} = S + p(\lambda, S, n)$.*

$\mathsf{KeyGen}(1^\lambda, 1^S)$**:**

- *Generate language and corresponding trapdoor* $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- *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* $(x, w) \leftarrow \mathsf{samp}\mathcal{L}(\mathsf{p})$.
- *Let* $\mathsf{k} \leftarrow \mathsf{Encap}'(\mathsf{pk}', x, w)$.
- *Let* $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m})$.
- *Return* $c = (x, c_{\mathsf{sym}})$.

$\mathsf{Dec}(\mathsf{sk}, c)$**:**

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

Correctness Follows from the correctness of (KeyGen', Encap', Decap', $\mathsf{Program}'$) and ($\mathsf{Enc}_{\mathsf{sym}}$, $\mathsf{Dec}_{\mathsf{sym}}$).

The security proof is similar but simpler than the CCA case.

Theorem 6 (Security)**.** *The PKE construction 4 has incompressible PKE security if (KeyGen', Encap', Decap', $\mathsf{Program}'$) is a programmable hash proof system with the listed parameters and ($\mathsf{Enc}_{\mathsf{sym}}$, $\mathsf{Dec}_{\mathsf{sym}}$) is an incompressible secure SKE with the listed parameters.*

Proof. We prove security via hybrids. First we list the hybrids and then argue their indistinguishability. In each hybrid we highlight the changes compared to the previous one.

$H_0(\lambda, S)$:

- Run key generation algorithm $\mathsf{KeyGen}(1^\lambda, 1^S)$ to obtain $(\mathsf{pk}, \mathsf{sk})$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ on public key pk to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$ \{0, 1\}$ uniformly at random.
- Run $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_b)$ to encrypt m_b.

- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_1 we explicitly represent what happens in KeyGen and Enc.

$H_1(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $\mathsf{pk} = \mathsf{pk}'$ and $\mathsf{sk} = \mathsf{sk}'$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1$ on public key pk to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$ \{0,1\}$ uniformly at random.
- Let $(x, w) \leftarrow \mathsf{samp}\mathcal{L}(\mathsf{p})$.
- Let $\mathsf{k} \leftarrow \mathsf{Encap}'(\mathsf{pk}', x, w)$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m}_b)$.
- Let $c = (x, c_{\mathsf{sym}})$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{pk}, c, \mathsf{st}_1)$ to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_2 we use the decapsulation mechanisms to encrypt the challenge message instead of encapsulation.

$H_2(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $\mathsf{pk} = \mathsf{pk}'$ and $\mathsf{sk} = \mathsf{sk}'$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ on public key pk to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$ \{0,1\}$ uniformly at random.
- Let $(x, w) \leftarrow \mathsf{samp}\mathcal{L}(\mathsf{p})$.
- Let $\mathsf{k} \leftarrow \mathsf{Decap}'(\mathsf{sk}', x)$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m}_b)$.
- Let $c = (x, c_{\mathsf{sym}})$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathsf{sk}}}(\mathsf{pk}, c, \mathsf{st}_1)$ to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_3 we sample x from $Y \subset X \setminus \mathcal{L}$ instead of $\mathcal{L}$.

$H_3(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^{\lambda}, 1^{n})$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $\mathsf{pk} = \mathsf{pk}'$ and $\mathsf{sk} = \mathsf{sk}'$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ on public key pk to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$\, \{0,1\}$ uniformly at random.
- Let $(x, \mathsf{td}_x) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$.
- Let $\mathsf{k} \leftarrow \mathsf{Decap}'(\mathsf{sk}', x)$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m}_b)$.
- Let $c = (x, c_{\mathsf{sym}})$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{pk}, c, \mathsf{st}_1)$ to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

In H_4 we program the secret key given to the adversary to decapsulate the ciphertext to the randomly chosen key k.

$H_4(\lambda, S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_{\mathcal{L}}) \leftarrow \mathsf{Gen}(1^{\lambda}, 1^{n})$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $\mathsf{pk} = \mathsf{pk}'$ and $\mathsf{sk} = \mathsf{sk}'$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1$ on public key pk to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$\, \{0,1\}$ uniformly at random.
- Let $(x, \mathsf{td}_x) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_{\mathcal{L}})$.
- Sample $\mathsf{k} \leftarrow\!\!\$\, \{0,1\}^{k(\lambda, S_{\mathsf{sym}}, n)}$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m})$.
- Let $c = (x, c_{\mathsf{sym}})$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{pk}, c)$ to produce a state st_2 smaller than S.
- Let $\mathsf{sk}'_{prog} \leftarrow \mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}', x, \mathsf{k})$.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk} = \mathsf{sk}'_{prog}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

$H_0 \approx H_1$:
The differences between H_0 and H_1 are purely syntactical. In H_1 we just show more detail of KeyGen and Enc.

$H_1 \approx H_2$:
In H_2 we merely change how the challenge ciphertext is calculated. By the correctness of the hash proof system these two hybrids look identical to the adversary.

$H_2 \approx_c H_3$:

In H_3 sample x from $Y \subset X \setminus \mathcal{L}$ instead of $\mathcal{L}$. These two hybrids are computationally indistinguishable by the language indistinguishability. Assume there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ that can distinguish the two hybrids H_2 and H_3 with a non-negligible advantage of ϵ. From this we construct a statistical adversary $\mathcal{A}'$ that can break language indistinguishability of the HPS with advantage ϵ.

$\mathcal{A}'(\mathsf{p}, x)$:

- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $\mathsf{pk} = \mathsf{pk}'$ and $\mathsf{sk} = \mathsf{sk}'$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ on public key pk to receive two messages m_0, m_1 and state st_1.
- Sample bit $b \leftarrow\!\!\$\, \{0,1\}$ uniformly at random.
- Let $\mathsf{k} \leftarrow \mathsf{Decap}'(\mathsf{sk}', x)$.
- Let $c_{\mathsf{sym}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}, \mathsf{m}_b)$.
- Let $c = (x, c_{\mathsf{sym}})$.
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{pk}, c, \mathsf{st}_1)$ to produce a state st_2 smaller than S.
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{sk}, \mathsf{st}_1, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$.
- The adversary wins if $b = b'$.

If $\mathcal{A}$ can distinguish H_2 from H_3 with advantage ε then the advantage of $\mathcal{A}'$ of distinguishing a x sampled from $\mathcal{L}$ and sampling x from $Y \subset X \setminus \mathcal{L}$ is also ϵ as it perfectly simulates H_2 in the case that $x \in \mathcal{L}$ and perfectly simulates H_3 in the other case.

$H_3 \approx_s H_4$:

According to programmable smoothness of $(\mathsf{KeyGen}', \mathsf{Encap}', \mathsf{Decap}')$ if $x \notin \mathcal{L}$ then $(\mathsf{pk}', \mathsf{sk}', x)$ is statistically close to $(\mathsf{pk}', \mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}', x, \mathsf{k}), x)$ for uniformly random k. Because this is exactly what we switch we get that H_3 and H_4 are statistically close.

$H_4 \approx \mathsf{Dist}^{\mathsf{IncomSKE}}_{\mathcal{A}', \Pi_{\mathsf{sym}}}$:

Finally, given an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ that wins experiment $H_4(\lambda, S)$ with probability ϵ we construct an adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2, \mathcal{A}'_3)$ that wins the $\mathsf{Dist}^{\mathsf{IncomSKE}}_{\mathcal{A}', \Pi_{\mathsf{sym}}}(\lambda, S + p(\lambda))$ experiment with probability ϵ.

$\mathcal{A}'_1(1^\lambda, 1^S)$:

- Generate language and corresponding trapdoor $(\mathsf{p}, \mathsf{td}_L) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$.
- Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}'(\mathsf{p})$.
- Let $\mathsf{pk} = \mathsf{pk}'$ and $\mathsf{sk} = \mathsf{sk}'$.
- Run the adversary $\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}_1 \leftarrow \mathcal{A}_1(\mathsf{pk})$ on public key pk to receive two messages m_0, m_1 and state st_1.
- Let $\mathsf{st}'_1 \leftarrow (\mathsf{pk}, \mathsf{sk}, \mathsf{td}_{\mathcal{L}}, \mathsf{st}_1)$.
- Return m_0, m_1, and st'_1

$\mathcal{A}'_2(\mathsf{st}'_1, c_{\mathsf{sym}})$:

- Parse $\mathsf{st}'_1 = (\mathsf{pk}, \mathsf{sk}, \mathsf{td}_{\mathcal{L}}, \mathsf{st}_1)$
- Let $(x, \mathsf{td}_x) \leftarrow \mathsf{samp}Y(\mathsf{p}, \mathsf{td}_L)$
- Let $c \leftarrow (x, c_{\mathsf{sym}})$
- Run the adversary $\mathsf{st}_2 \leftarrow \mathcal{A}_2(\mathsf{pk}, c, \mathsf{st}_1)$ to produce a state st_2 smaller than S
- Return state $\mathsf{st}'_2 \leftarrow (x, \mathsf{td}_x, \mathsf{st}_2)$ smaller than $S_{\mathsf{sym}} = S + p(\lambda)$

$\mathcal{A}'_3(\mathsf{k}, \mathsf{st}'_1, \mathsf{st}'_2, \mathsf{m}_0, \mathsf{m}_1)$:

- Parse $\mathsf{st}_1 = (\mathsf{pk}, \mathsf{sk} = \mathsf{sk}', \mathsf{td}_{\mathcal{L}}, \mathsf{st}'_1)$

- Parse $\mathsf{st}_2 = (x, \mathsf{td}_x, \mathsf{st}'_2)$
- Program $\mathsf{sk}'_{prog} \leftarrow \mathsf{Program}(\mathsf{td}_{\mathcal{L}}, \mathsf{td}_x, \mathsf{sk}', x, \mathsf{k})$
- Run the final adversary $b' \leftarrow \mathcal{A}_3(\mathsf{pk}, \mathsf{sk}'_{prog}, \mathsf{st}_2, \mathsf{m}_0, \mathsf{m}_1)$
- Return b'

$\mathcal{A}'$ wins $\mathsf{Dist}^{\mathsf{IncomSKE}}_{\mathcal{A}',\Pi_{\mathsf{sym}}}(\lambda, S + p(\lambda))$ iff $\mathcal{A}$ wins in $H_4(\lambda, S)$ because $\mathcal{A}'$ perfectly simulates H_4 from the perspective of $\mathcal{A}$.

□