Optimal Threshold Traitor Tracing

Sourav Das[1], Pratish Datta[2], Aditi Partap[*3], Swagata Sasmal[*4], and Mark Zhandry[2,3]

[1] Category Labs
[2] NTT Research
[3] Stanford University
[4] Indian Statistical Institute

Abstract. Threshold encryption distributes decryption capability across n parties such that any t of them can jointly decrypt a ciphertext, while smaller coalitions learn nothing. However, once t or more parties collude, traditional threshold schemes provide no accountability: a coalition of t or more parties can pool its keys into a pirate decoder that enables unrestricted decryption, all without any risk of being exposed. To address this, Boneh, Partap, and Rotem [CRYPTO '24] introduced *threshold traitor tracing* (TTT), which equips threshold encryption with traceability. Yet, all known TTT schemes either suffer from parameter sizes growing with at least $n^{1/3}$, or rely on indistinguishability obfuscation to achieve optimal parameters.

In this paper, we present the *first* TTT schemes with *optimal parameters*, where public keys, secret keys, and ciphertexts are all bounded by $\mathsf{poly}(\lambda, \log n)$, built solely from standard cryptographic tools and assumptions. Our first construction relies on the decisional Bilinear Diffie–Hellman (DBDH) assumption in prime order bilinear groups. Our second scheme is a candidate construction based on the Learning with Errors (LWE) assumption, which relies on the existence of secret sharing schemes with certain properties. This construction is plausibly post-quantum secure, and supports ramp-thresholds where decryption requires a larger coalition than those tolerated by security. Both of our constructions provide traceability against coalitions of arbitrary sizes.

To achieve these results, we introduce a new primitive, *Attribute-Based Threshold Encryption* (ABTE), which generalizes both threshold and attribute-based encryption. We then combine ABTE with Mixed Functional Encryption through a new compiler to obtain our TTT schemes. We believe ABTE is a powerful primitive that may have independent applications beyond optimal TTT.

⋆ This work was done while Aditi Partap and Swagata Sasmal were interns at NTT Research.

Table of Contents

Optimal Threshold Traitor Tracing 1
Sourav Das, Pratish Datta, Aditi Partap, Swagata Sasmal*, and Mark Zhandry*
1 Introduction 3
2 Technical Overview 5
2.1 A generic compiler for TTT 5
2.1.1 TTT from Threshold PLBE 6
2.1.2 Threshold PLBE from ABTE + mixed FE. 7
2.2 Constructing (Key Policy) ABTE 9
2.2.1 ABTE from pairings. 10
2.2.2 Ramp ABTE from Lattices. 11
2.3 Putting it all together 13
2.3.1 Optimal TTT from pairings. 14
2.3.2 Optimal ramp-threshold traitor tracing from Lattices. 14
2.4 Additional Related Work 14
2.5 Outline 16
3 Definitions 16
3.1 Threshold Traitor Tracing 16
3.1.1 Extensions 19
3.2 Mixed-Functional Encryption 20
4 Attribute-based Threshold Encryption 21
5 Attribute-based Threshold Encryption from Pairings 23
5.1 Preliminaries 24
5.2 Our Construction 24
6 Attribute-based Threshold Encryption from Lattices 28
6.1 Preliminaries on Lattices 28
6.2 Preliminaries on Ramp Secret Sharing 30
6.3 Our Candidate Construction 32
7 Threshold Traitor Tracing from Threshold PLBE 38
7.1 Threshold Private Linear Broadcast Encryption 38
7.2 A generic compiler for TTT from Threshold PLBE 41
7.2.1 Decoder-based Threshold PLBE from 1-bounded Threshold PLBE. 41
7.2.2 TTT from Decoder-based Threshold PLBE. 42
8 Threshold PLBE from ABTE and mixed FE 43
9 Risky and Limited Threshold Traitor Tracing from Weak Threshold PLBE 47
9.1 Achieving Weak Decoder-based Indistinguishability from 0-bounded Indistinguishability 48
9.2 Constructing Risky and Limited TTT from Weak TPLBE 49
10 Eliminating Riskiness and Limitedness in Tracing 50
10.1 Risk Mitigation Compiler 50
10.2 Limitedness Mitigation Compiler 52

1 Introduction

Threshold cryptography [35, 39, 36, 33] is a paradigm for distributing trust in cryptography. A t-out-of-n threshold encryption scheme [56, 23, 48, 28, 41, 63, 18, 49, 10, 66] distributes key shares among n parties such that any coalition of at least t parties can jointly decrypt a well-formed ciphertext, where as any coalition of $t-1$ or less parties learns nothing about the underlying message. Owing to this balance between robustness and privacy, threshold encryption has emerged as a foundational primitive with diverse applications, including secure electronic voting [29], privacy-preserving sealed-bid auctions [61], and, more recently, the design of encrypted mempools in blockchain ecosystems [9, 59, 34, 62].

Threshold Traitor Tracing. Threshold encryption, while secure against collusion of up to $t-1$ parties, offers no accountability once t or more parties collude. If such a coalition sells its key shares to an adversary, the adversary gains full decryption capability, rendering the system completely insecure. In traditional threshold schemes, the colluding parties face no risk, since there is no mechanism to hold them accountable for their misbehavior.

To address this gap, Boneh, Partap, and Rotem [14] introduced the notion of *threshold traitor tracing* (TTT), which augments threshold encryption with traceability. They consider the scenario where a coalition of t or more parties uses their keys to build a *decoder box*—a device or program (conceived as an obfuscated algorithm or tamper-proof hardware) that takes a ciphertext as input and outputs its decryption. In addition to the standard *semantic security* guarantee, the defining property of TTT is *traceability*: a designated tracer must be able to efficiently identify at least one colluding party by interacting with the decoder box only through black-box access. This accountability mechanism deters parties from selling their key shares, since doing so exposes them to the risk of being traced. For the problem to be meaningful, it is further essential that the system parameters – public keys, ciphertexts, and key shares – remain compact[1].

The State-of-The-Art. Boneh, Partap, and Rotem [14] gave the first formalization of TTT and presented four constructions, each with different efficiency tradeoffs. One construction has constant-size public keys and ciphertexts, but requires quadratic-size ($O(n^2)$) secret keys for each party. Another has constant-size secret keys, but at the cost of larger public keys and ciphertexts of size $O(n^{1/3})$. More recently, several follow-up works [19, 22, 11] have advanced this line of research, proposing new TTT schemes with improved tradeoffs. Nevertheless, all known constructions still suffer from at least polynomial parameter sizes in n, with the exception of an indistinguishability obfuscation (iO) based scheme that achieves optimal parameters i.e. $\mathsf{poly}(\log n)$, as observed in [14]. (A detailed comparison appears in Section 2.4 and Table 1.) This state of affairs naturally leaves open a central and tantalizing question:

Open Problem. *Can we design threshold traitor tracing schemes with optimal parameters, namely, where the sizes of public parameters, secret keys, and ciphertexts grow only poly-logarithmically in the number of users, while relying solely on simple, well-founded cryptographic assumptions and tools?*

[1] One can trivially achieve traceability by using threshold encryption with $O(n)$-sized public keys and ciphertexts, but such schemes are impractical. See [14, §7] for details.

Our Results

We resolve the above open question by presenting the first threshold traitor tracing (TTT) schemes with *optimal parameters*, i.e., the sizes of public parameters, secret keys, and ciphertexts are $\mathsf{poly}(\lambda, \log n)$ for security parameter λ, while only relying on standard cryptographic tools. Our contributions are twofold:

An Optimal Pairing-based Construction. Our first result is an optimal TTT scheme based on prime-order bilinear groups. We prove its security under the decisional bilinear Diffie–Hellman (DBDH) assumption.

Theorem 1 (Informal). *Under the DBDH assumption, there exists an optimal threshold traitor tracing scheme in which public parameters, secret keys, and ciphertexts all have size* $\mathsf{poly}(\lambda, \log n)$.

A candidate post-quantum ramp-threshold construction. Our second contribution is a candidate ramp-threshold traitor tracing scheme based on the Learning with Errors (LWE) assumption, thereby offering plausibly post-quantum security. In this setting, semantic security is guaranteed against coalitions of up to t_p parties, while decryption requires at least $t_c > t_p$ parties. Note that, even though coalitions larger than t_p may be able to break semantic security, traceability is guaranteed against any coalition of parties that produces a decoder box. This candidate construction can be instantiated with any ramp-threshold secret sharing scheme which satisfies the following properties: (i) linearity, (ii) small reconstruction coefficients, (iii) constant-size secret shares and (iv) share uniformity (see Section 2.2.2 for details). When instantiated with a secret sharing scheme with all the above properties[2], our candidate construction can provide the first plausibly post-quantum construction of ramp TTT with optimal parameters.

Theorem 2 (Informal). *Under the LWE assumption, there exists a candidate ramp-threshold traitor tracing scheme in which public parameters, secret keys, and ciphertexts all have size* $\mathsf{poly}(\lambda, \log n)$. *This scheme can be instantiated with any linear secret sharing scheme that achieves share uniformity, reconstruction coefficients that scale with* $\mathsf{poly}(\lambda, n)$ *and share sizes that scale with* $\mathsf{poly}(\lambda, \log n)$.

We provide a detailed comparison of our TTT constructions with prior works in Section 1, and describe our high level ideas next.

Our approach. The technical core of our work is a new generic compiler for TTT, built around the new *Attribute-Based Threshold Encryption* (ABTE) primitive we introduce. ABTE generalizes threshold encryption and attribute-based encryption, i.e., it enables attribute-based access control on top of threshold encryption. Concretely, each party is issued a key associated with a policy and an index, while ciphertexts are labeled with attributes. Decryption succeeds only if a coalition of at least t parties comes together, all holding keys whose policies accept the ciphertext's attribute. Security requires that even if up to $(t-1)$ parties with accepting policies collude – along with any number of parties holding non-accepting policies – they learn nothing about the plaintext. We believe ABTE is a broadly useful primitive, with applications beyond TTT.

Our notion of ABTE allows us to generalize earlier traitor tracing compilers [17, 46, 67, 68] from non-threshold to the threshold setting. Specifically, we show how to combine ABTE with *Mixed*

[2] We are not aware of any impossibility results on the existence of secret sharing schemes with all these properties. We leave the task of constructing such a scheme as future work.

Functional Encryption [46, 68] via a sequence of compilers to obtain TTT. While the compilers are a natural generalization of prior work, the threshold setting introduces subtleties in the security analysis that we address.

Theorem 3. *Assuming there exists a (i) mixed functional encryption (mFE) scheme, and an (ii) ABTE scheme for policies that can compute the mFE decryption circuit, there exists a threshold traitor tracing scheme (TTT).*

We then provide two instantiations of ABTE. Our first construction, based on the Decisional Bilinear Diffie–Hellman (DBDH) assumption, supports arithmetic span programs as policies. This enables us to combine it with the pairings-based mFE scheme of [68] to obtain our first optimal TTT scheme from pairings. Our second construction is a candidate scheme based on Learning with Errors (LWE) and supports arbitrary circuit policies. However, it only achieves a ramp-threshold guarantee. The gap stems from the need for a linear secret sharing scheme with (i) constant-size secret shares, to achieve optimality, (ii) small reconstruction coefficients, to control noise growth in the lattice setting, in addition to (iii) share uniformity, for security. Since threshold secret sharing with even the first two properties simultaneously is not known, we resort to a ramp-threshold secret sharing scheme. Instantiating our compiler with this ramp ABTE together with the LWE-based mFE scheme of [46] yields our second result. Both our ABTE constructions are realized through a semi-generic framework that integrates existing attribute-based encryption schemes with threshold secret-sharing techniques, thereby combining fine-grained access control with distributed decryption resilience.

Applications of ABTE beyond TTT. We view ABTE as a fundamental primitive whose utility extends well beyond threshold traitor tracing. Just as threshold encryption enhances the security of public-key encryption by distributing the decryption, ABTE strengthens attribute-based encryption (ABE) by enabling threshold decryption in the attribute-based setting. This opens the door to natural applications where both fine-grained access control and protection against data misuse are crucial. For example, in the context of encrypting medical records, ABE can ensure that only authorized doctors and researchers may access particular patient data [1, 4]. In this scenario, ABTE allows us to prevent misuse of data by any single entity by requiring collaboration among at least t authorized users i.e. doctors or researchers, before any record can be decrypted.

2 Technical Overview

We now give an overview of our techniques. We split it into three parts. Section 2.1 presents our generic compiler for constructing TTT. In Section 2.2, we describe our pairings-based and LWE-based constructions of ABTE. Lastly, in Section 2.3, we briefly discuss how we instantiate our compiler with known mixed FE constructions and our ABTE schemes to achieve our results.

2.1 A generic compiler for TTT

We present our generic compiler in two steps. In Section 2.1.1, we show how to generically construct TTT from a new primitive called Threshold Private Linear Broadcast Encryption (PLBE)[3]. In

[3] We remark that [14] also present a TTT construction based on the specific PLBE scheme of [44]. In contrast, our approach demonstrates how to *generically* realize TTT from *any* threshold PLBE, thereby providing a broader and more modular foundation.

	Assumptions	Public-Key size	Secret-Key size	Ciphertext size	Perfect threshold
Boneh et al. [14, §5.2]	Pairings	$O(1)$	$O(n^2)$	$O(1)$	Yes
Boneh et al. [14, App. B]	Pairings	$O(n^{1/3})$	$O(1)$	$O(n^{1/3})$	Yes
Bormet et al. [19, §6.2]	Pairings, GGM	$O(n^2)$	$O(1)$	$O(1)^{\dagger}$	Yes
Bormet et al. [19, §7]	Pairings, GGM	$O(n^4)$	$O(1)$	$O(1)$	Yes
Bhattacharyya et al [11]	Pairings	$O(1)$	$O(n^2)$	$O(1)$	Yes
Canard et al [22]	Pairings	$O(n^4)$	$O(n^2)$	$O(n^2)$	Yes
This work (Thm. 1)	Pairings	$O(1)$	$O(1)$	$O(1)$	Yes
This work(Thm. 2)‡	LWE	$O(1)$	$O(1)$	$O(1)$	Ramp

Table 1. Comparing the efficiency of Threshold Traitor Tracing (TTT) constructions. All the numbers are for full collusion-resistance, meaning that the tracer must find a traitor even if all n parties collude to construct the decoder box. The tracing algorithm for all the schemes runs in time $\mathsf{poly}(N)$, which is sufficient since this algorithm only needs to be executed when tracing decoders. Note that we ignore $\mathsf{poly}(\lambda)$ factors and assume $\log(n) < \lambda$.

† This is amortized over $O(n)$ ciphertexts.

‡ This is a candidate construction that can be instantiated with any ramp-threshold secret sharing scheme that satisfies linearity, share uniformity, constant-size shares and small reconstruction coefficients (see Section 2.2.2 for details).

Section 2.1.2, we present a generic construction of Threshold PLBE from Attribute-based Threshold encryption and Mixed Functional Encryption. Note that there are some subtleties in applying these compilers in the pairings setting; we expand on this in Section 2.3.

2.1.1 TTT from Threshold PLBE We start with some background on an existing compiler for (non-threshold) traitor tracing (TT), which is a special case of TTT with threshold $t = 1$.

Background on Traitor Tracing from PLBE. We start by describing a compiler for constructing (non-threshold) traitor tracing from Private Linear Broadcast Encryption (PLBE), given in [17]. A PLBE system for n parties has a public key pk, master secret key msk, and secret keys of all parties $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n$. Messages in PLBE can be either encrypted in a *public-key mode* or a *secret-key mode*. The public-key mode encryption (which we also refer to as the normal encryption) requires only the public-key pk and the underlying message, and any sk_j can decrypt such public-key mode ciphertexts. To encrypt in a secret-key mode, the secret-key encryption algorithm, EncSK, takes as input msk, a message and an index i, and outputs a ciphertext that only can be decrypted using secret keys sk_j for parties $j > i$. For security, PLBE require three properties. The first is *index-hiding*, which requires that an adversary with keys of all parties except i, cannot distinguish between secret-key encryptions to index $i-1$ and i. The second is *message-hiding*, which requires that secret-key encryptions of two different messages m_0, m_1 to index n must be indistinguishable, even if the adversary has all n keys. Lastly, *indistinguishability* requires that a normal encryption of message m created using pk is indistinguishable from a EncSK encryption of m to the index 0, even if the adversary has all n keys.

To build traitor tracing, we run the PLBE setup and give each PLBE secret key sk_i to party i. To trace a decoder box D, the tracing algorithm first measures (with several samples) the probability that D correctly decrypts a ciphertext encrypted to index i for all $i \in [0, n]$. If the box D decrypts

public-key ciphertexts with probability ϵ, then it must decrypt ciphertexts encrypted to index 0 with roughly the same probability, by indistinguishability. Moreover, D cannot decrypt encryptions to index n, by message-hiding. Hence, there must exist some index $i \in [n]$ where D's success probability drops significantly between encryptions to index $i-1$ and i. By index-hiding, this gap implies D must contain sk_i; hence, the tracer simply outputs i as a colluder.

Our TTT compiler. To adapt the above approach to the threshold setting, we need to *thresholdize* the decryption process such that a ciphertext can only be decrypted by a set of t or more parties, while still supporting tracing. Our key observation is that decryption in the PLBE-based TT scheme is identical to the PLBE decryption. Thus, if we can thresholdize PLBE decryption, we can obtain threshold traitor tracing using a similar compiler.

Based on this observation, we define a new primitive, which we refer to as threshold PLBE or TPLBE. In a t-out-of-n TPLBE, the setup algorithm outputs $\mathsf{pk}, \mathsf{msk}$ along with n keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$. Unlike classic PLBE, here a ciphertext encrypted under pk requires at least t users to decrypt: each user produces a partial decryption using its key, and any t partial decryptions can be combined to recover the message. Similarly, the EncSK algorithm produces ciphertexts with respect to an index i, which can only be decrypted by t users with indices greater than i. We require four security properties. The first three are indistinguishability, index-hiding, and message-hiding, which are defined as above. Crucially, the threshold does not play a role in these properties. The fourth requirement is standard semantic security. This requires that an adversary that corrupts up to $t-1$ parties, must not learn anything about the encrypted message. This must hold even if the adversary is allowed to see partial decryptions on valid ciphertexts of its choice.

With TPLBE in place, we can lift the PLBE-to-TT compiler directly to the threshold setting. Setup simply distributes TPLBE keys to users. Decryption proceeds as in TPLBE: each party contributes a decryption share, and any t shares can be combined. The tracing algorithm is also identical as above. To see why this is a secure TTT scheme, first observe that semantic security of TTT follows directly from semantic security of TPLBE. Moreover, the proof of traceability extends naturally from the PLBE case since the indistinguishability, index-hiding, and message-hiding properties remain intact. In terms of efficiency, observe that the public key, secret key and ciphertext sizes for the resulting TTT scheme are equal to the corresponding sizes for the TPLBE scheme. Hence, with this compiler, the task of constructing optimal TTT reduces to the task of constructing optimal TPLBE.

2.1.2 Threshold PLBE from ABTE + mixed FE.

We begin by reviewing the existing compiler that constructs PLBE from the combination of Attribute-Based Encryption (ABE) and mixed Functional Encryption (mFE).

Background on PLBE from ABE + mFE. We describe the compiler given in [46] for constructing PLBE from Attribute-based encryption (ABE) and mixed function encryption (mFE). An mFE scheme supports two encryption modes: (i) public-key (or normal) encryption, which takes only the public key and outputs a ciphertext ct, and (ii) secret-key encryption, which takes the master secret msk and a function f to produce a ciphertext ct_f. Decryption keys are tied to messages: a normal ciphertext ct will decrypt to 1 when using the key sk_m for any message m. Additionally, decrypting a secret-key ciphertext ct_f using sk_m will output $f(m)$. mFE security requires a) *accept-indistinguishability*, which states that normal ciphertexts are indistinguishable from secret-key ciphertexts for a function f, unless one has a secret key for message m such that $f(m) \neq 1$,

and b) *function-indistinguishability*, which requires that an adversary learns nothing more about a function f except for the values $f(m)$ for messages m whose keys are in its possession.

To build PLBE, [46] combine mFE with (key-policy) ABE in the following way. The messages are encrypted using ABE, with the attribute being set to an mFE ciphertext. In the public-key mode, the attribute is a normal mFE ciphertext ct. Next, to encrypt to index i in the secret-key mode, we set the ABE attribute to $\mathsf{ct}_{\mathsf{comp}_i}$, an mFE secret-key ciphertext encrypting the comparison function comp_i that outputs 1 only if the input is $> i$. Each party j receives an ABE secret key with the policy being the mFE decryption circuit hard-wired with the mFE secret key sk_j corresponding to index j. Clearly, any party can decrypt a normal ciphertext since the policy for any key evaluates to one on ct by mFE correctness. On the other hand, decrypting $\mathsf{ct}_{\mathsf{comp}_i}$ only outputs one for parties $j > i$, so only these parties can decrypt the ciphertext encrypted to i, as desired.

Note that while mFE can be defined for arbitrary function classes, an mFE for comparisons comp_i is enough for building PLBE (and hence TT). At the same time, the policy class supported by ABE must be expressive enough to capture the mFE decryption circuit.

Our TPLBE compiler. To construct a threshold PLBE using the approach above, we need to *thresholdize* decryption so that ciphertexts can only be decrypted by t or more parties. Towards this goal, we observe that the decryption algorithm in the PLBE construction above is precisely the decryption algorithm of the ABE scheme. Therefore, it suffices to thresholdize the ABE decryption algorithm. To achieve this, we introduce the primitive of Attribute-based Threshold Encryption (ABTE), which we describe next.

Attribute-based Threshold Encryption. In a t-out-of-n (key-policy) ABTE, each user's secret key is tied both to a policy and to the user's index. Specifically, the key for user i and policy $C_i : \mathcal{X} \to 0, 1$ is denoted sk_{i,C_i}. As in standard ABE, encryption of a message m is associated with a public attribute $x \in \mathcal{X}$.

The crucial difference from ABE lies in the decryption process – a ciphertext labeled with attribute x can only be decrypted by a coalition of t parties whose policies are all satisfied by x. Concretely, each party can only compute a partial decryption share using its key, and the message can be recovered only by combining t such shares from parties with satisfying policies i.e. $C_i(x) = 1$. Thus, ABTE generalizes ABE and threshold encryption: it enforces fine-grained access control via attributes, while also distributing decryption authority across multiple parties. Note that our ABTE notion is not to be confused with threshold ABE [50], wherein the policies themselves are restricted to be threshold predicates, or with registered, distributed or multi-authority ABE where the setup process is decentralized. We discuss these related notions in Section 2.4.

In terms of security, ABTE captures both the security of ABE and the standard semantic security of threshold encryption. Specifically, for an ABTE to be secure, we require that an adversary cannot learn anything about a message encrypted with attribute x^*, even if it can query (i) an unbounded number of keys for parties with policies not satisfied by x^* (as in ABE security), (ii) keys of up to $t-1$ parties with satisfying policies, and (iii) decryption shares for valid ciphertexts of its choice (as in threshold encryption security).

The compiler. With the notion of ABTE in place, we are ready to describe the compiler to construct TPLBE using mFE and ABTE. Conceptually, the compiler is identical to the one described earlier: each party is assigned the ABTE key corresponding to its index i together with the same policy as before, and encryption proceeds in the same manner. The only modification lies in the decryption procedure: each party generates a partial decryption share, and any collection of t valid shares can be combined to recover the message. For security, semantic security of the resulting

TPLBE can be reduced directly to semantic security of the ABTE scheme, since the adversary is only allowed to query keys of up to $t-1$ parties. The remaining three PLBE security properties—indistinguishability, index-hiding, and message-hiding are proven exactly as for the PLBE compiler.

Regarding efficiency, note that the public key, ciphertext, and secret key sizes of the resulting TPLBE scheme depend directly on the corresponding sizes for ABTE and mFE. Since efficient mFE schemes (with $\mathsf{poly}(\log(n), \lambda)$ parameters) for comparison functions are known from both pairings [68] and LWE [46], the main efficiency challenge lies in constructing efficient ABTE schemes, which we will describe in the next section.

2.2 Constructing (Key Policy) ABTE

Recall that ABTE generalizes ABE by thresholdizing the decryption process. In ABTE, each party i receives a secret key sk_{i,C_i} corresponding to index i and policy C_i. A ciphertext labeled with attribute x can be decrypted only if a subset of at least t parties, each with a satisfying policy (i.e., $C_i(x) = 1$), collaborates. Intuitively, a party i whose policy is satisfied recovers only a partial decryption of the message using its key sk_{i,C_i}, and t or more such partial decryptions together suffice to recover the plaintext.

Naive attempts. We begin with two natural but flawed approaches to constructing ABTE. A first idea is to secret-share the ABE master secret key among n parties. While this enforces that at least t parties are needed for decryption, it provides no mechanism to enforce the policies: any set of t parties can reconstruct the master secret key and decrypt any ciphertext, regardless of attributes.

A second idea is to generate a secret key for a policy (as in ABE) and then secret-share this key among multiple parties. However, this does not allow assigning distinct policies to different parties, since the secret key that is shared across them corresponds to a single policy.

Our technique in a nutshell. Based on the above failed attempts, we observe that each party must be issued a single secret key that simultaneously encodes *both* its policy and its index. We achieve this by observing that in many existing ABE schemes, the master key pair contains a *masking term* such that, encryption hides the message by combining it with a randomized version of this masking term. Parties are assigned secret keys with respect to the masking term, such that they can recover the randomized masking term in the ciphertext only if their policy is satisfied by the attribute.

Our central idea is to *thresholdize* the masking term. Instead of a single global masking term, we first secret-share it into n shares using a t-out-of-n secret-sharing scheme. Note that encryption still uses the global masking term. Each party i then receives a secret key derived from the ABE key-generation algorithm, but instantiated with respect to the i-th share of the masking term. This way, party i can recover its share of the randomized masking term only when its policy is satisfied by the ciphertext's attribute (and obtains no useful information otherwise). Once t parties with satisfying policies have recovered their respective shares, they can reconstruct the full masking term via the secret-sharing reconstruction algorithm, and from it, recover the underlying message. As we will see later in this section, for this idea to work, the secret-sharing scheme must be compatible with the ABE structure, so that its reconstruction procedure meshes correctly with the ABE decryption process.

We now describe two instantiations of this idea: one based on bilinear pairings (Section 2.2.1), and the other based on LWE (Section 2.2.2). Our LWE-based construction only supports ramp

thresholds. Our approach is general and may be applied to other ABE schemes, and we leave further instantiations as an interesting direction for future work.

2.2.1 ABTE from pairings. Recall that to construct Threshold PLBE from ABTE and mFE using our compiler, the ABTE scheme must support the mFE decryption circuit as part of its policy class. Hence, we thresholdize the pairings-based ABE of Ishai and Wee [52], as it supports arithmetic span programs as policies and can therefore capture the decryption circuit of the only known mFE from pairings [68]. We first provide a brief background on this ABE construction, and then explain how we use our ideas above to obtain an ABTE. Our scheme uses asymmetric bilinear groups, but we present the overview using symmetric pairings for simplicity.

Background on Ishai-Wee ABE [52]. Let g be a generator of a symmetric bilinear group of order q, and let $\mathcal{X} := \mathbb{Z}_q^z$ denote the space of attributes. The master secret key is then $\mathsf{msk} := (\alpha, a, \boldsymbol{v})$ for uniformly random $\alpha, a \leftarrow\!\$\, \mathbb{Z}_q$ and $\boldsymbol{v} := [v_1, \ldots, v_z] \leftarrow\!\$\, \mathbb{Z}_q^z$. The master key pair is as follows:

$$\mathsf{mpk} := (g^a, g^{v_1}, \ldots, g^{v_z}, e(g,g)^\alpha) \quad \mathsf{msk} := (\alpha, a, \boldsymbol{v}).$$

To encrypt a message μ with respect to an attribute $\boldsymbol{x} := [x_1, \ldots, x_z] \in \mathbb{Z}_q^z$, sample $s \leftarrow\!\$\, \mathbb{Z}_q$, and output

$$\mathsf{ct}_{\boldsymbol{x}} := (\mathsf{ct}_0 := g^s \ , \ \{\mathsf{ct}_i := g^{(ax_i + v_i)s}\}_{i\in[z]} \ , \ \hat{\mathsf{ct}} := e(g,g)^{\alpha s} \cdot \mu).$$

Here, the terms $\{\mathsf{ct}_i\}_{i\in[z]}$ in the ciphertext encode the attribute $\boldsymbol{x}$ in the exponent, s is the randomness and α acts as the masking term (as defined above).

To generate the key for a party with an arithmetic span program $V = \{(\boldsymbol{y}_j, \boldsymbol{z}_j)\}_{j\in[m]}$ where $\boldsymbol{y}_j, \boldsymbol{z}_j \in \mathbb{F}_q^\ell$ and $\rho : [m] \to [z]$ as the policy, we sample a vector $\boldsymbol{u} \leftarrow\!\$\, \mathbb{Z}_q^\ell$ whose last element is equal to α. Then, the key is set to:

$$\mathsf{sk} := \left\{ g^{\langle \boldsymbol{u}, \boldsymbol{y}_j\rangle \cdot a^{-1}}, g^{\langle \boldsymbol{u}, \boldsymbol{z}_j\rangle - (v_{\rho(j)} \cdot \langle \boldsymbol{u}, \boldsymbol{y}_j\rangle)\cdot a^{-1}} \right\}_{j\in[m]}.$$

Intuitively, the secret key encodes α in the exponent with respect to the span program. This encoding can be combined with the ciphertext components $\mathsf{ct}_0, \ldots, \mathsf{ct}_z$ to recover the randomized masking term $e(g,g)^{\alpha s}$, but only if the attribute x satisfies the program. Dividing $\hat{\mathsf{ct}}$ by this term then reveals the message.

Our ABTE Construction. To thresholdize this ABE, we generate t-out-of-n secret shares of the masking term α. Let $(\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_q^n$ be these shares. Then, we generate the secret key $\mathsf{sk}_{i,(V,\rho)}$ for party i for a policy (V, ρ) with respect to the i-th share α_i (instead of α), i.e., we use a random a vector $\boldsymbol{u} \leftarrow\!\$\, \mathbb{Z}_q$ whose last element is equal to α_i (instead of α). This implies that if the attribute $\boldsymbol{x}$ satisfies the policy (V, ρ), then party i would be able to recover its decryption share $e(g,g)^{\alpha_i \cdot s}$ by combining its key with the ciphertext, using the Ishai-Wee decryption algorithm. Finally, any t decryption shares can be combined by running the reconstruction algorithm of the secret-sharing scheme in the exponent to recover $e(g,g)^{\alpha \cdot s}$ and decrypt the ciphertext.

This scheme can be shown to satisfy CPA security. Intuitively, an adversary holding at most $t-1$ keys cannot recover the masking term of the challenge ciphertext, since these keys correspond to only $t-1$ shares of α, which remain uniformly distributed due to the privacy of the secret-sharing scheme. Furthermore, keys associated with non-satisfying policies reveal nothing, due to the security of the underlying ABE scheme. Moreover, an adversary that queries decryption shares on arbitrary ciphertexts only obtains terms of the form $e(g,g)^{\alpha_i \cdot s}$ (where s is the randomness of the

ciphertext). These terms do not help decrypt the challenge ciphertext, as the challenge ciphertext will be encrypted using a different randomness.

We now discuss how to instantiate the secret-sharing scheme. For decryption, we need to reconstruct the secret in the exponent, hence the reconstruction algorithm must be linear. Moreover, for our resulting ABTE construction to be efficient, the secret shares of α must be small, as the secret key size of each party scales with the share size. For these reasons, we use Shamir secret-sharing.

We present the formal construction in Section 5.

2.2.2 Ramp ABTE from Lattices. To build a lattice-based ABTE, we use the key-policy ABE scheme by Boneh et al. [13] for circuits as our starting point. We start with background on this ABE, followed by our construction.

Background on Boneh et al.'s ABE. The public key for this ABE scheme consists of matrices $\boldsymbol{B}_1, \ldots, \boldsymbol{B}_z, \boldsymbol{D} \leftarrow\$ \mathbb{Z}_q^{\ell \times m}$, where z is the length of the attribute vector, and $m = \Theta(\ell)$. We also sample a matrix $\boldsymbol{A} \in \mathbb{Z}_q^{\ell \times m}$ uniformly randomly along with a trapdoor. To encrypt a message μ with respect to an attribute $x \in \mathbb{Z}_q^z$, we use a variant of dual Regev encryption [60, 42] using the following matrix as the public key:

$$\boldsymbol{H}_x = \boldsymbol{A} \mid x_1\boldsymbol{G} + \boldsymbol{B}_1 \mid \ \ldots \ \mid x_z\boldsymbol{G} + \boldsymbol{B}_z$$

More formally, the ciphertext is computed as follows:

$$\mathsf{ct}_x := \left(\mathsf{ct}_{x,0} := \boldsymbol{H}_x^\top \boldsymbol{s} + \boldsymbol{e} \ , \ \mathsf{ct}_{x,1} := \boldsymbol{D}^\top \boldsymbol{s} + \boldsymbol{e}_1 + \lceil q/2 \rceil \cdot \mu\right),$$

where $\boldsymbol{e}, \boldsymbol{e}_1$ are sampled from a gaussian distribution, and $\boldsymbol{s}$ is sampled uniformly randomly from $\mathbb{Z}_q^\ell$. The secret key for a party with respect to a circuit $f : \mathbb{Z}_q^z \to \mathbb{Z}_q$ as its policy is the dual-Regev decryption key with respect to the matrix $\boldsymbol{A}|\boldsymbol{B}_f$. Here, $\boldsymbol{B}_f$ serves as the encoding of the circuit f, and is uniquely determined by $\boldsymbol{B}_1, \ldots, \boldsymbol{B}_z$ and f. More formally, the secret key for f is a short pre-image $\boldsymbol{R}_f$ of $\boldsymbol{D}$ with respect to $\boldsymbol{A}|\boldsymbol{B}_f$:

$$(\boldsymbol{A} \mid \boldsymbol{B}_f)^\top \boldsymbol{R}_f = \boldsymbol{D}.$$

To decrypt a ciphertext for attribute x that satisfies the policy f i.e. $f(x) = 0$, [13] show that it is possible to efficiently transform ct_x into a dual-Regev encryption with respect to the public key $(\boldsymbol{A} \mid f(x) \cdot \boldsymbol{G} + \boldsymbol{B}_f)$, using knowledge of just f and x. More formally, we can compute:

$$\mathsf{ct}_{x,f} := (\boldsymbol{A} \mid f(x) \cdot \boldsymbol{G} + \boldsymbol{B}_f)^\top \boldsymbol{s} + \boldsymbol{e}'$$

where $\boldsymbol{s}$ is the same vector sampled during encryption, and $\boldsymbol{e}'$ is a short vector derived from $\boldsymbol{e}$. Now, if $f(x) = 0$, then the new public key is $\boldsymbol{A}|\boldsymbol{B}_f$, and the secret key $\mathsf{sk}_f = \boldsymbol{R}_f$ is a decryption key for exactly this key. In more detail, we can get $\boldsymbol{D}^\top \boldsymbol{s}$ (with some noise) by computing $\boldsymbol{R}_f^\top \cdot \mathsf{ct}_{x,f}$. Then, decryption can be done by simply subtracting this from $\mathsf{ct}_{x,1}$ and rounding. More formally,

$$\mu = \mathsf{Round}\left(\mathsf{ct}_{x,1} - \boldsymbol{R}_f^\top \cdot \mathsf{ct}_{x,f}\right).$$

Since both $\boldsymbol{R}_f^\top$ and the noise in $\mathsf{ct}_{x,f}$ are short, decryption does indeed output the correct message.

Our candidate ramp ABTE Construction. To apply our secret-sharing technique to this ABE, we observe that the matrix $\boldsymbol{D}$ can be seen as the masking term that is used to hide the message

during encryption. Following our central idea described above, we can secret-share the matrix $\boldsymbol{D}$, and compute the secret key of the i-th party for policy f as a short pre-image of the i-th share of $\boldsymbol{D}$ with respect to $(\boldsymbol{A}|\boldsymbol{B}_f)$. In more detail, let us use $\boldsymbol{D}_1, \ldots, \boldsymbol{D}_n$ to denote the t-out-of-n secret shares of $\boldsymbol{D}$ based on some secret sharing scheme (we will discuss how to instantiate the secret sharing scheme in the next paragraph). Then, the key sk_{i,f_i} for party i and policy f_i will be a short matrix $\boldsymbol{R}_{i,f_i}$ such that:

$$(\boldsymbol{A} \mid \boldsymbol{B}_{f_i})^\top \boldsymbol{R}_{i,f_i} = \boldsymbol{D}_i.$$

For decryption, any single party with a satisfying policy can now only recover a share of the masking term $\boldsymbol{D}^\top \boldsymbol{s}$ (with small noise). More formally, party i with policy f_i will first transform the ciphertext ct_x into an encryption ct_{x,f_i} with respect to public key $(\boldsymbol{A} \mid f_i(x) \cdot \boldsymbol{G} + \boldsymbol{B}_{f_i})$, as in [13]. Then, if $f_i(x) = 0$, it is easy to see that $\boldsymbol{R}_{i,f_i}^\top \mathsf{ct}_{x,f_i}$ is equal to $\boldsymbol{D}_i^\top \boldsymbol{s} + \boldsymbol{e}_i$ where $\boldsymbol{e}_i$ is short – this is a noisy share of the masking term $\boldsymbol{D}^\top \boldsymbol{s}$. Hence, party i simply outputs this as its decryption share. Any t such decryption shares can be combined to reconstruct $\boldsymbol{D}^\top \boldsymbol{s}$ with some added noise, which is sufficient to decrypt the ciphertext.

Instantiating the secret sharing scheme. Let us now see what properties we want from a secret sharing scheme to instantiate our approach. Clearly, to recover noisy $\boldsymbol{D}^\top \boldsymbol{s}$ from noisy $\boldsymbol{D}_i^\top \boldsymbol{s}$ values, the reconstruction algorithm of the secret sharing scheme must be linear. Moreover, for achieving optimal secret key size for the resulting ABTE, the size of each secret share must be independent of the number of parties. While Shamir secret sharing satisfies both these properties, using Shamir will lead to inefficient parameters. To understand why, let us dive into the decryption process. Recall that party i with a satisfying policy can recover $\boldsymbol{D}_i^\top \boldsymbol{s} + \boldsymbol{e}_i$ for some short noise $\boldsymbol{e}_i$. To reconstruct the masking term i.e. $\boldsymbol{D}^\top \boldsymbol{s}$ from decryption shares of any t-sized set $S \subseteq [n]$, in Shamir secret sharing, we compute:

$$\Sigma_{i \in S} \lambda_{i,S} \cdot (\boldsymbol{D}_i^\top \boldsymbol{s} + \boldsymbol{e}_i) = \boldsymbol{D}^\top \boldsymbol{s} + \Sigma_{i \in S} \lambda_{i,S} \cdot \boldsymbol{e}_i,$$

where $\{\lambda_{i,S}\}$ are the Lagrange coefficients for reconstruction. Unfortunately, these coefficients, when interpreted as elements in $\mathbb{Z}_q$, are large and therefore, blow up the total noise when multiplied with $\boldsymbol{e}_i$ terms. This would imply that, to ensure correct decryption, we would have to scale the modulus q with $n!$, making the public key, ciphertext and secret keys grow with $O(n)$, which is far from optimal. To avoid this blowup, the secret sharing scheme must satisfy a third property, that the reconstruction coefficients must be small.

Additionally, for the security proof, we need the secret sharing scheme to satisfy share uniformity and linearity. Share uniformity is a stronger notion that secrecy which requires that the shares of any unauthorized subset of parties are distributed uniformly at random. Linearity requires that both the sharing and reconstruction algorithms of the scheme are linear. To see why we need both these properties, let us dive deeper into the security proof. Recall that the adversary can query keys of upto $t-1$ parties with satisfying policies – let us consider a particular query (i, f_i) where $f_i(x^*) = 0$ and x^* is the challenge attribute. When reducing to LWE, this means that the reduction needs to sample a short matrix $\boldsymbol{R}_{i,f_i}$ such that $(\boldsymbol{A}|\boldsymbol{B}_{f_i})^\top \boldsymbol{R}_{i,f_i}$ equals the ith share $\boldsymbol{D}_i$ of the matrix $\boldsymbol{D}$. Since the reduction will not have the trapdoor for the LWE matrix $\boldsymbol{A}$, one way to generate such keys is to sample $\boldsymbol{R}_{i,f_i}$ from a gaussian distribution, and then set $\boldsymbol{D}_i$ to be $(\boldsymbol{A}|\boldsymbol{B}_{f_i})^\top \boldsymbol{R}_{i,f_i}$. This will mean that the ith share $\boldsymbol{D}_i$ is distributed uniformly at random, for each of the $t-1$ key queries. Hence, the secret sharing scheme must satisfy share uniformity to ensure that these shares are distributed exactly as in the real game. Additionally, the reduction will need to compute the

remaining $n-(t-1)$ shares of $\boldsymbol{D}$ that are consistent with the $t-1$ shares sampled in this way. A secret sharing scheme with linear sharing naturally supports this. To summarize, instantiating our approach requires a secret sharing scheme with four properties: (i) linearity, (ii) share uniformity, (iii) small reconstruction coefficients and (iv) constant-size secret shares.

Prior works have relied on linear secret sharing schemes with binary reconstruction coefficients, also referred to as $\{0,1\}$-LSSS [12, 20, 24, 37, 27]. While these schemes achieve linearity and binary reconstruction coefficients, they suffer from two limitations: (i) the share sizes grow with atleast $\Omega(n^{1.5})$ and (ii) they do not satisfy share uniformity (see [8] for details).

Hence, we consider the scheme given in [6], which constructs secret sharing schemes from LDPC erasure codes. Their scheme achieves linearity, small reconstruction coefficients as well as constant-size shares. It can also achieve share uniformity provided the underlying erasure code satisfies certain properties (see Section 6.2 for details). While we are currently not aware of codes that satisfy these properties, we conjecture that such codes do indeed exist and leave the task of constructing such codes as an open problem. We note that this scheme only supports the *ramp-threshold* access structure, which is characterized by two parameters τ_p and τ_c. Privacy is guaranteed for up to τ_p corruptions, but reconstruction requires at least $\tau_c > \tau_p$ shares. Instantiating our approach with this secret sharing scheme (using codes that satisfy certain properties) will result in an attribute-based ramp-threshold encryption scheme with optimal parameter sizes. We leave the task of designing ABTE with perfect threshold from plausibly post-quantum-secure assumptions as a future direction.

Achieving CPA security. Recall that our notion of CPA security allows the the adversary to see decryption shares any valid ciphertext of its choice. An avid reader may observe that the scheme described above does not achieve this notion. The problem is, every time the adversary gets a decryption share of party i, it learns something about the secret key of that party. In more detail, recall that the decryption share is simply the inner product of the secret key $\boldsymbol{R}_{i,f_i}$ with the ciphertext ct_{x,f_i}. Hence, the adversary can easily learn the secret key of this party by querying its decryption shares for a polynomial number of ciphertexts. This issue also comes up in other lattice-based threshold cryptosystems (see [12, 37, 30] and the references therein). We resolve this by using the standard technique of adding additive noise to the decryption share, i.e.:

$$d_i := \boldsymbol{R}_{i,f_i}^{\top}\mathsf{ct}_{x,f_i} + \mathsf{noise}.$$

This is also known as noise smudging or flooding [12]. Note that, for the noise to completely hide the inner product, it must come from a wide distribution. This requires us to scale the modulus q with $\mathsf{exp}(\lambda)$, but this does not worsen the parameters because the underlying ABE scheme [13] already requires q to grow exponentially with λ for security.

We present the full scheme in Section 6.3.

2.3 Putting it all together

In this section, we describe how we can instantiate our TTT compiler using our ABTE constructions and known mFE schemes, to achieve our results.

[4] We use Limited to denote that the tracer can only trace boxes which can decrypt with high (but constant) probability. This is traditionally referred to as 'threshold' traitor tracing (see [68]), but we rename it to Limited to avoid confusion with our notion of threshold.

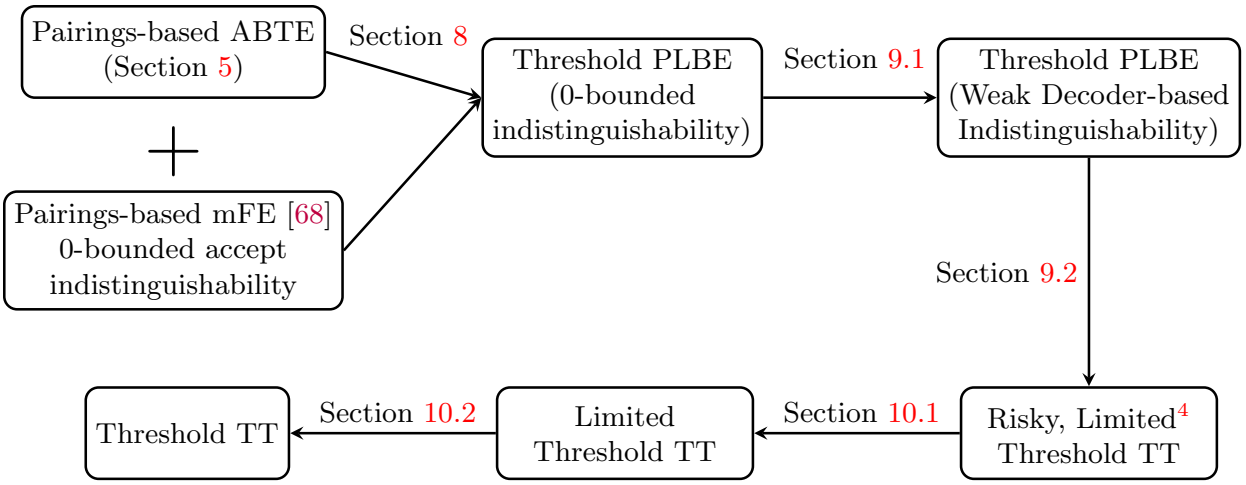

Fig. 1. The sequence of compilers for constructing our pairings-based TT scheme with optimal parameters. The definitions of 0-bounded accept indistinguishability for mFE, q-bounded indistinguishability, Weak Decoder-based indistinguishability for TPLBE, and risky, Limited TT can be found in Sections 3.2 and 9, and 3.1 respectively.

2.3.1 Optimal TTT from pairings. For optimal TTT from pairings, we instantiate the TPLBE compiler with the mFE scheme by Zhandry [68]. This mFE scheme relies on the decisional BDH assumption and its decryption circuit can be expressed as an arithmetic branching program over a large field. Since our pairings-based ABTE scheme supports arithmetic span programs as policies (which includes arithmetic branching programs), we can use it along with the mFE to get a TPLBE scheme, and apply the TTT compiler to get our first result.

Unfortunately, there is a flaw in the approach described above. At a high level, Zhandry's mFE only satisfies a weak notion of security. Consequently, the resulting TPLBE after applying the first compiler also achieves weak security – unfortunately, the TPLBE to TTT compiler cannot be applied to such a TPLBE scheme. To solve this issue, we observe that [68] also deal with the same issue to achieve their optimal TT in the non-threshold setting. We show how to adapt their techniques to construct a different sequence of compilers which can be used to obtain TTT even from an mFE with weak security. Figure 1 gives a summary of this sequence of compilers. Although each of the compilers is a natural generalization of the ones given in [68, 67], there are some subtleties in the security analysis when adapting them to the threshold setting.

In terms of efficiency, each of these compilers only grow the parameters by a factor of $\mathsf{poly}(\lambda)$. Hence, applying these compilers to Zhandry's mFE and our pairings-based ABTE results in a pairings-based TTT with optimal parameters.

2.3.2 Optimal ramp-threshold traitor tracing from Lattices. We instantiate our TPLBE compiler with the LWE-based mFE scheme from [46] and our LWE-based ramp-threshold ABTE. By applying the TTT compiler on the resulting TPLBE scheme, we achieve our second result given in Theorem 2. In Figure 2, we summarize the compilers we apply to achieve this result.

2.4 Additional Related Work

Threshold Traitor tracing. Following the introduction of threshold traitor tracing by [14], there have been many followups to construct TTT with stronger security properties:

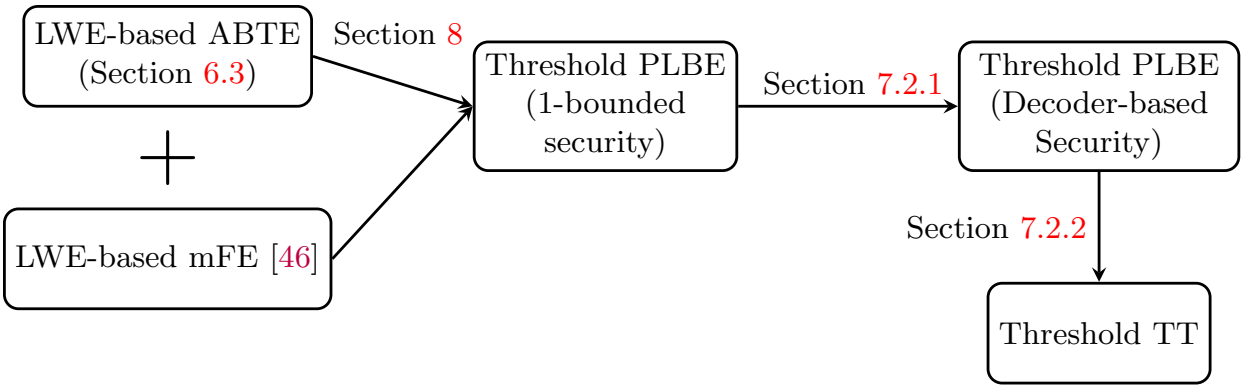

Fig. 2. The sequence of compilers for constructing our LWE-based ramp-threshold TT scheme with optimal parameters. The q-bounded and decoder-based notions of security for TPLBE are formalized in Section 7.1.

Public traceability. Our schemes only achieve secret tracing, wherein tracing requires the master secret key. [19, 22] present TTT constructions with public traceability, wherein the tracing algorithm does not require any secret key. But the parameters for their constructions grow with at least n^2. An interesting question is then how to obtain publicly traceable threshold traitor tracing systems from pairings (or lattices) with optimal parameters.

CCA security. Our constructions of TTT only achieve CPA security. [11] construct a TTT scheme with CCA security, but the secret-key size for their scheme grows with $O(n^2)$. We leave the task of designing TTT schemes with optimal parameters and CCA security as a future direction.

Silent setup. In a concurrent work, Chakrabarti et al. [25] construct TTT schemes with silent setup, meaning that every party can run the key generation algorithm locally, without interacting with the other parties. In other words, there is no need for a trusted setup or a complex distributed key generation protocol. But their parameters are far from optimal – the ciphertext size scales with $\sqrt{n}$. Moreover, they rely on pairings and prove security in the generic group model. Designing TTT schemes with optimal parameters as well as silent setup is an interesting direction for future research.

Related ABE notions. [50] construct an ABE scheme which supports threshold predicate as the policies, i.e. a key can decrypt a ciphertext only if the ciphertext has at least a threshold number of satisfying attributes. This is different from our notion of ABTE, wherein we thresholdize the decryption process, and put no restrictions on the policy class. Similarly, [40] define and constructed attribute-based signcryption for threshold policies.

Other related notions of ABE include Registered and Distributed ABE [51, 7], wherein there is no central authority holding the master secret key for issuing keys. Instead, users can generate secret keys on their own and then register the associated public key with a "key curator" along with their attributes. In Multi-Authority ABE [54, 26, 57, 64], the central authority is decentralized – the keys for different attributes are issued by different authorities. These are different from our notion of ABTE, where we thresholdize the decryption process (note that we still have a master secret key that is used to generate keys for all parties). An interesting future direction would be to study registered or multi-authority versions of ABTE.

Traceability for other threshold primitives. [47] and [15] introduced the notion of traceability in secret sharing, where a less-than-threshold number of share holders may sell their shares to an

adversary. As explained in [14], it is unclear if techniques from these works can be used to construct TTT. [16] study traceability in the context of threshold verifiable random functions, wherein a coalition of less-than-threshold parties may sell their key shares. In contrast, TTT focuses on a scenario where a set of t or more parties sell their shares in the form of a decryption box. [38] generalize the notion of traitor tracing to functional encryption. We leave the task of thresholdizing this notion as future work.

2.5 Outline

The rest of the paper is organized as follows. We formally define Threshold Traitor Tracing (TTT) and mixed Functional Encryption (mFE) in Section 3. We then introduce Attribute-based Threshold Encryption (ABTE) in Section 4. We present our pairings-based and lattice-based ABTE constructions in Sections 5 and 6 respectively. In Section 7, we formally define Threshold PLBE (TPLBE) and then present our generic compiler to construct TTT from TPLBE. In Section 8, we show how to generically construct TPLBE from ABTE and mixed FE. Lastly, Sections 9 and 10 present the additional compilers required in the pairings setting (due to some subtleties, as mentioned in Section 2.3).

3 Definitions

3.1 Threshold Traitor Tracing

A threshold traitor tracing scheme is a tuple of five polynomial-time algorithms $\mathsf{TTT} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Combine}, \mathsf{Trace})$:

- $\mathsf{KeyGen}(1^\lambda, 1^n, 1^t) \to (\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{tk})$ is the probabilistic **key generation algorithm**. It takes in the security parameter $\lambda \in \mathbb{N}$, the number n of decryptors and the threshold t. It outputs a public key pk, a combiner public key pkc, secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$, along with the tracing key tk.
- $\mathsf{Enc}(\mathsf{pk}, \mu) \to \mathsf{ct}$ is the probabilistic **encryption algorithm**. It takes as input the public key pk and a message μ, and it outputs a ciphertext ct. We may use $\mathsf{Enc}(\mathsf{pk}, \mu; r)$ to refer to a deterministic encryption algorithm, which takes as input a randomness r sampled from a randomness space $\mathcal{R}$.
- $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) \to d_i$ is the deterministic **decryption algorithm**. It takes in a decryptor's secret key sk_i and a ciphertext ct, and its output is a decryption share d_i of ct under sk_i.
- $\mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}, \{d_j\}_{j \in \mathcal{J}}) \to \mu/\perp$ is the deterministic **combiner algorithm**. Its input is a combiner public key pkc, a ciphertext ct, a subset $\mathcal{J}$ of $[n]$, and decryption shares $\{d_j\}_{j \in \mathcal{J}}$. It outputs either a message μ or a rejection symbol $\perp$.
- $\mathsf{Trace}^{D(\cdot)}(\mathsf{pk}, \mathsf{tk}, 1^{1/\epsilon}, \mu_0, \mu_1) \to \mathcal{J}$ is a PPT **tracing algorithm** that takes as input the public key pk, the tracing key tk, an error bound $\epsilon = \epsilon(\lambda)$ and messages μ_0, μ_1. Trace also has oracle access to a *"decoder"* algorithm D. It outputs a subset $\mathcal{J} \subseteq [n]$.

Correctness. An honestly generated ciphertext should be correctly decrypted by any subset of t decryptors. We formalize this notion in Definition 1 below.

Definition 1. *A threshold traitor tracing scheme* TTT *is said to be correct if, there exists a negligible function* $\mathsf{negl}(\lambda)$ *of the security parameter* λ*, such that, for all* $\lambda \in \mathbb{N}$*, all* $t < n \in \mathbb{N}$*, all* t*-sized subsets* $\mathcal{J} \subseteq [n]$*, and all messages* μ *in the message space, it holds that*

$$\Pr\left[\mu' = \mu \ : \begin{array}{c} (\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{tk}) \leftarrow\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^t) \\ c \leftarrow\!\$\ \mathsf{Enc}(\mathsf{pk}, \mu) \\ \forall\ i \in \mathcal{J},\ d_i \leftarrow \mathsf{Dec}(\mathsf{sk}_i, c) \\ \mu' \leftarrow \mathsf{Combine}(\mathsf{pkc}, c, \mathcal{J}, \{d_j\}_{j\in\mathcal{J}}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

CPA Security. We require that a threshold traitor tracing scheme must satisfy the notion of semantic security, as is standard for threshold decryption. Roughly speaking, we require that an adversary must not be able to learn anything about the encrypted message, even if it can corrupt upto $t-1$ parties. We also allow the adversary to query partial decryptions of *valid* ciphertexts of its choice. We formalize this by giving the adversary access to a decryption oracle, which takes as input a message μ, randomness r, along with index i. The challenger computes an encryption of μ using r, and outputs the partial decryption of this ciphertext with respect to party i's key[5]. Note that this is stronger than the CPA definition considered in [14], where there was no decryption oracle, and stronger than the one considered in [31] since we allow the adversary to choose the randomness r for the ciphertext, when querying partial decryptions. The full security game and definition is given in Fig. 3 and Definition 2.

Semi-adaptive adversaries. The security games as defined in Figure 3 allow for fully-adaptive adversaries, in the sense that they do not pose any restrictions on the order in which the adversary decides on its oracle queries. Proving security against such adaptive adversaries is known to be a challenging task, already for threshold encryption (without tracing) [31]. Since the problem of adaptive security is not the focus of this work, we focus on a semi-adaptive version of security, wherein the adversary must make all the key queries before it makes any partial decryption query. This is captured in the game by maintaining a bit b which is set to 1 if the adversary queries the partial decryption oracle. The secret-key oracle checks whether this bit has been set, and if so, it will ignore the query, returning $\perp$. Otherwise, it will continue as defined in Fig. 3. We denote this setting as semi-adaptive, or *sa*.

Definition 2 (CPA security). *We say that* TTT *satisfies semi-adaptive CPA security if for every probabilistic polynomial time adversary* $\mathcal{A}$*, the following function is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{sa-cpa}}_{\mathcal{A},\mathsf{TTT}}(\lambda) := \left|\frac{1}{2} - \Pr[\mathbf{IND\text{-}CPA}^{sa}_{\mathcal{A},\mathsf{TTT}}(\lambda) = 1]\right|.$$

Traceability. Informally, if the decoder D, given a ciphertext c, can learn any information about the encrypted message, then Trace traces D back to at least one party from the coalition that "manufactured" it. Moreover, it should not falsely accuse any honest party. The following definition captures this property using the security experiment in Fig. 4.

Definition 3. *Let* $\epsilon = \epsilon(\lambda)$ *and* $\delta = \delta(\lambda)$ *be functions of the security parameter* $\lambda \in \mathbb{N}$*. A threshold traitor-tracing scheme* $\mathsf{TTT} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Combine}, \mathsf{Trace})$ *with key space* $\mathcal{K} = \{\mathcal{K}_\lambda\}$ *is* (ϵ, δ)*-secure if for every* PPT *adversary* $\mathcal{A}$ *and for every* $\lambda \in \mathbb{N}$*, the following two conditions hold*

[5] This is weaker than CCA security since we do not allow the adversary to ask for partial decryptions for arbitrary ciphertexts. We leave the task of formalizing and achieving CCA security as a future direction.

Experiment $\mathbf{IND\text{-}CPA}_{\mathcal{A},\mathsf{TTT}}(\lambda)$

$\mathcal{J} \leftarrow \emptyset$

$(n, t, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$

$(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{tk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^t)$

$(\mu_0, \mu_1, \mathsf{state}) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{sk}(\cdot),\mathsf{dec}(\cdot,\cdot,\cdot)}(\mathsf{state}, \mathsf{pk}, \mathsf{pkc}, \mathsf{tk})$

$b \leftarrow\!\!\$\ \{0,1\},\ c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, \mu_b)$

$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{sk}(\cdot),\mathsf{dec}(\cdot,\cdot,\cdot)}(\mathsf{state}, c)$

if $|\mathcal{J}| \geq t$ **then return** 0

if $b' = b$ **then return** 1 **else return** 0

Oracle $\mathsf{sk}(i)$

if $i \notin [n]$ **then return** $\perp$

$\mathcal{J} \leftarrow \mathcal{J} \cup \{i\}$

return sk_i

Oracle $\mathsf{dec}(\mu, r, i)$

if $i \notin [n]$ **then return** $\perp$

return $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{Enc}(\mathsf{pk}, \mu; r))$

Fig. 3. The semantic security experiment for a threshold traitor tracing scheme TTT and an adversary $\mathcal{A}$.

Experiment $\mathbf{ExpTrace}_{\mathcal{A},\mathsf{TTT},\epsilon}(\lambda)$

$\mathcal{J} \leftarrow \emptyset$

$(n, t, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$

$(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{tk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^t)$

$(D, \mu_0, \mu_1) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{sk}(\cdot)}(\mathsf{state}, \mathsf{pk}, \mathsf{pkc})$ // output a decoder alg. D

$\mathcal{J}' \leftarrow\!\!\$\ \mathsf{Trace}^{D(\cdot)}(\mathsf{pk}, \mathsf{tk}, 1^{1/\epsilon(\lambda)}, \mu_0, \mu_1)$ // trace decoder

return $(\mathsf{pk}, D, \mathcal{J}, \mathcal{J}')$

Fig. 4. The tracing experiment for a threshold traitor tracing scheme TTT and an adversary $\mathcal{A}$. The corruption oracle $\mathsf{sk}(\cdot)$ is defined as in Fig. 3.

$$\Pr[\mathsf{GoodTr}] \geq \Pr[\mathsf{GoodDec}] - \delta(\lambda) \qquad \textit{and} \qquad \Pr[\mathsf{BadTr}] \leq \delta(\lambda)$$

where $(pk, D, \mathcal{J}, \mathcal{J}') \leftarrow\!\$\ \mathbf{ExpTrace}_{\mathcal{A},\mathsf{TTT},\epsilon}(\lambda)$ *as defined in Figure 4. The events* GoodDec, GoodTr, *and* BadTr *are defined as follows:*

- GoodDec *occurs when* $P(D) \geq 1/2 + \epsilon(\lambda)$, *where* $P(D)$ *is defined by*

$$P(D) := \Pr\big[D(c) = b \ : \ b \leftarrow\!\$\ \{0,1\},\ c \leftarrow\!\$\ \mathsf{Enc}(pk, m_b)\big].$$

- GoodTr *holds when* $\mathcal{J}' \neq \emptyset$ *and* $\mathcal{J}' \subseteq \mathcal{J}$.
- BadTr *holds when* $\mathcal{J}' \neq \emptyset$ *and* $\mathcal{J}' \not\subseteq \mathcal{J}$.

We say that TTT *is* secure *if there exists a negligible function* $\nu = \nu(\lambda)$ *such that* TTT *is* $(1/p, \nu)$*-secure for every polynomial* $p = p(\lambda)$. *For an adversary* $\mathcal{A}$ *and a TTT scheme* TTT, *we define*

$$\mathsf{Adv}^{\mathrm{tt}}_{\mathcal{A}_1,\mathsf{TTT}}(\lambda) := \max\big\{\Pr[\mathsf{GoodDec} = 1] - \Pr[\mathsf{GoodTr} = 1],\ \ \Pr[\mathsf{BadTr} = 1]\big\}.$$

Secret Tracing Key. Note that in our definition of tracing security, the tracing key is kept secret from the adversary. This is because, for our constructions, knowledge of this key can allow the adversary to evade tracing. But, semantic security is preserved even against adversaries with access to this key. This is captured in our Strong-CPA definition, wherein the adversary is given access to the tracing key, along with upto $t-1$ secret keys.

3.1.1 Extensions We define the following variations/extensions of the TTT definition:

Ramp-threshold Traitor Tracing. A (τ_p, τ_c) ramp-threshold traitor tracing (for $0 < \tau_p < \tau_c \leq 1$) is defined as above, but we require correctness to hold only for subsets of $[n]$ of size $\geq \tau_c \cdot n$, and we require Strong-CPA to hold only for up to $\tau_p \cdot n$ corruptions. The traceability definition remains the same as before.

Risky Threshold Traitor Tracing. We define a risky threshold traitor tracing scheme similar to [45]; a traitor is only accused with some small but noticeable probability. Let $\alpha = \alpha(n, \lambda)$ be a polynomial.

Definition 4. *A threshold traitor tracing scheme* TTT *is* α*-risky if, for all* PPT *adversaries* $\mathcal{A}$ *and all inverse-polynomials* ϵ, *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that* $\Pr[\mathsf{BadTr}] \leq \mathsf{negl}(\lambda)$ *and* $\Pr[\mathsf{GoodTr}] \geq \alpha \cdot \Pr[\mathsf{GoodDec}] - \mathsf{negl}(\lambda)$.

Limited Threshold Traitor Tracing. A Limited scheme [58] traitor tracing is one where a malicious user is accused only for very good decoders that succeed a constant fraction of the time. We generalize this notion for threshold traitor tracing below. Note that in the traitor tracing literature, this is traditionally referred to as 'threshold' traitor tracing, but we rename it to Limited to avoid confusion with threshold decryption as in our work.

Definition 5. *A threshold traitor tracing scheme* TTT *is* Limited secure *if there exists a constant* $\epsilon \in (0, 1/2)$ *such that, for all* PPT *adversaries* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that* $\Pr[\mathsf{BadTr}] \leq \mathsf{negl}(\lambda)$ *and* $\Pr[\mathsf{GoodTr}] \geq \Pr[\mathsf{GoodDec}] - \mathsf{negl}(\lambda)$.

In the case of Limited secure schemes, the constant ϵ is hard-coded into the algorithm Trace, and we omit ϵ as an input to Trace. A TTT scheme that is δ-risky and ϵ-Limited may also be referred to as a (ϵ, δ)-risky Limited TTT scheme.

3.2 Mixed-Functional Encryption

Consider function classes $\mathcal{F} = \{\mathcal{F}_\lambda\}_\lambda$ and message spaces $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$, where $f : \mathcal{M}_\lambda \to \{0,1\}$ for each $f \in \mathcal{F}_\lambda$. A mixed functional encryption scheme Mixed-FE [46], for function classes $\mathcal{F}$ and message spaces $\mathcal{M}$, consists of five PPT algorithms $\mathsf{MFE} = (\mathsf{Setup}, \mathsf{EncSK}, \mathsf{EncPK}, \mathsf{KeyGen}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{mpk}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ, and outputs the public parameters pp, the public key mpk and the master secret key msk. The public parameters pp will be an implicit input to all other algorithms.
- $\mathsf{KeyGen}(\mathsf{msk}, \mu) \to \mathsf{sk}_\mu$. The key generation algorithm takes as input master secret key msk and a message $m \in \mathcal{M}_\lambda$. It outputs a secret key sk_μ.
- $\mathsf{EncPK}(\mathsf{mpk}) \to \mathsf{ct}$. The normal encryption algorithm takes as input the master public key mpk, and outputs a ciphertext ct.
- $\mathsf{EncSK}(\mathsf{msk}, f) \to \mathsf{ct}$. The secret key encryption algorithm takes as input master secret key msk and a function $f \in \mathcal{F}_\lambda$. It outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}_\mu, \mathsf{ct}) \to \{0,1\}$. The decryption algorithm takes as input a secret key sk_μ and a ciphertext ct, and it outputs a single bit.

Definition 6 (Correctness). *A mixed functional encryption scheme* MFE *is said to satisfy correctness if there exists negligible functions* negl_1, negl_2 *such that for all* $\lambda \in \mathbb{N}$, *and for all inputs* $\mu \in \mathcal{M}_\lambda$ *and all functions* $f \in \mathcal{F}_\lambda$,

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_\mu, \mathsf{ct}) = 1 \;:\; \begin{array}{c}(\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ \mathsf{sk}_\mu \leftarrow\!\$\ \mathsf{KeyGen}(\mathsf{msk}, \mu) \\ \mathsf{ct} \leftarrow\!\$\ \mathsf{EncPK}(\mathsf{mpk})\end{array}\right] \geq 1 - \mathsf{negl}_1(\lambda)$$

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_\mu, \mathsf{ct}) = f(m) \;:\; \begin{array}{c}(\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ \mathsf{sk}_\mu \leftarrow\!\$\ \mathsf{KeyGen}(\mathsf{msk}, \mu) \\ \mathsf{ct} \leftarrow\!\$\ \mathsf{EncSK}(\mathsf{mpk}, f)\end{array}\right] \geq 1 - \mathsf{negl}_2(\lambda)$$

q-Bounded Security. The security of an MFE scheme is captured through the notions of *function indistinguishability* and *accept indistinguishability* in the presence of an adversary that makes at most q many secret key encryption queries and unbounded number of key generation queries. The first property states that for any two functions $f_0, f_1 \in \mathcal{F}_\lambda$, if the output of the two functions is identical on every μ such that sk_μ is in the adversary's possession, then any PPT adversary cannot distinguish between the secret key encryptions of f_0 and f_1. The *accept indistinguishability* property states that the secret key encryption and public key encryption of the always accepting function should be indistinguishable. Moreover, we only need to consider a weaker "*restricted*" notion where the adversary needs to declare the two challenge functions $f_0, f_1 \in \mathcal{F}_\lambda$ at the beginning of the security game and must submit all its q encryption queries before submitting any key generation query. We formally re-state the two properties defined in [46] for completeness.

Definition 7 (q-bounded Restricted Function Indistinguishability). *Let* $q(\cdot)$ *be any fixed polynomial. A Mixed-FE scheme* MFE *is said to satisfy* q*-bounded selective function indistinguishability if there exists a negligible function* negl *such that for all* $\lambda \in \mathbb{N}$, *and for any* PPT *adversary*

$\mathcal{A}$,

$$\Pr\left[b' = b \;:\; \begin{array}{c} f_0, f_1 \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ b \leftarrow\!\$\ \{0,1\} \\ \mathsf{ct} \leftarrow\!\$\ \mathsf{EncSK}(\mathsf{msk}, f_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{EncSK}(\mathsf{msk},\cdot)}(\mathsf{ct}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where

- $\mathcal{A}$ *can make at most* $q(\lambda)$ *many queries to* $\mathsf{EncSK}(\mathsf{msk}, \cdot)$ *and,*
- f_0 *and* f_1 *agree on every input* μ *submitted by* $\mathcal{A}$ *to* $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$*, i.e.* $f_0(\mu) = f_1(\mu)$*, and*
- $\mathcal{A}$ *must make all (at most* $q(\lambda)$*)* $\mathsf{EncSK}(\mathsf{msk}, \cdot)$ *oracle queries before making any query to* $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ *oracle.*

Definition 8 (q-bounded Restricted Accept Indistinguishability). *A Mixed-FE scheme* MFE *is said to satisfy* q*-bounded selective accept indistinguishability if there exists a negligible function* negl *such that for all* $\lambda \in \mathbb{N}$*, and for any* PPT *adversary* $\mathcal{A}$*,*

$$\Pr\left[b' = b \;:\; \begin{array}{c} f^* \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ b \leftarrow\!\$\ \{0,1\} \\ \mathsf{ct}_0 \leftarrow\!\$\ \mathsf{EncPK}(\mathsf{mpk}); \mathsf{ct}_1 \leftarrow\!\$\ \mathsf{EncSK}(\mathsf{msk}, f^*) \\ b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{EncSK}(\mathsf{msk},\cdot)}(\mathsf{ct}_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where

- $\mathcal{A}$ *can make at most* $q(\lambda)$ *many queries to* $\mathsf{EncSK}(\mathsf{msk}, \cdot)$ *and,*
- *every query* μ *that* $\mathcal{A}$ *submits to* $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ *should satisfy* $f^*(\mu) = 1$ *as well as* $f(\mu) = 1$ *for every query* f *that* $\mathcal{A}$ *makes to* $\mathsf{EncSK}(\mathsf{msk}, \cdot)$
- $\mathcal{A}$ *must make all (at most* $q(\lambda)$*)* $\mathsf{EncSK}(\mathsf{msk}, \cdot)$ *oracle queries before making any query to* $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ *oracle.*

Remark 1. Our use of q here is off by 1 from [68], who set q to be the total number of ciphertexts given to the adversary, including the single "challenge" ciphertext. We instead use q as the number of additional ciphertexts given to the adversary, on top of the challenge ciphertext, to be consistent with [46].

4 Attribute-based Threshold Encryption

A (key-policy) Attribute-based Threshold Encryption (ABTE) scheme for a family of predicate classes $\mathcal{C} = \{\mathcal{C}_\lambda\}$ is a tuple of PPT algorithms $\mathsf{ABTE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Combine})$:

- $\mathsf{Setup}(1^\lambda, 1^n, 1^t, 1^z) \to (\mathsf{mpk}, \mathsf{pkc}, \mathsf{msk})$ is the probabilistic **setup** algorithm that takes as inputs the security parameter $\lambda \in \mathbb{N}$, the number of decryptors n, the threshold t, the attribute length parameter $z \in \mathbb{N}$, and outputs the master public key mpk, the combining public key pkc and the master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, i, C) \to \mathsf{sk}_C^i$ is the probabilistic **key generation** algorithm that takes as inputs an index i, the master secret key msk, a predicate $C \in \mathcal{C}_\lambda$ representing a policy, and outputs a secret key sk_C^i for index i with respect to this policy. For any attribute x, we use $C(x) = 1$ to denote that x satisfies this predicate.

– $\mathsf{Enc}(\mathsf{mpk}, \mu, x) \to \mathsf{ct}_x$ is the probabilistic **encryption** algorithm that takes as inputs the master public key mpk, the plaintext message μ, an attribute x and outputs a ciphertext ct_x. We may use $\mathsf{Enc}(\mathsf{mpk}, \mu, x; r)$ to denote the deterministic version of this algorithm, which takes as input randomness r samples from a randomness space $\mathcal{R}$.
– $\mathsf{Dec}(\mathsf{sk}^i_C, \mathsf{ct}_x) \to d_i$ is the probabilistic **decryption** algorithm that takes as input a secret key sk^i_C for some index i and policy C, a ciphertext ct_x generated with respect to attribute x and outputs the partial decryption d_i of ct_x under sk^i_C.
– $\mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}_x, \{d_j\}_{j\in\mathcal{S}}) \to \mu$ is the deterministic **combiner** algorithm that takes as inputs a set of decryption shares $\{d_j\}_{j\in\mathcal{S}}$ for a set $\mathcal{S} \subseteq [n]$ of size t and outputs a message μ.

Correctness. An ABTE scheme ABTE must satisfy correctness, which requires that any subset of t parties must be able to decrypt an honestly generated ciphertext for any attribute, as long as the attribute satisfies the policy associated with the keys of *all* the t parties. Definition 9 formalizes this notion.

Definition 9 (Correctness). *An Attribute-based Threshold encryption scheme* $\mathsf{ABTE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Combine})$ *is correct if, for all* $\lambda \in \mathbb{N}$*, all* $t \le n \in \mathbb{N}$*, all* $m, \ell, z \in \mathbb{N}$*, all attributes* $x \in \mathbb{F}_q^z$*, all messages* μ*, all subsets* $\mathcal{S}$ *of size* t*, and corresponding policies* $\{C_i\}_{i\in\mathcal{S}}$ *such that* x *satisfies* C_i *for all* $i \in \mathcal{S}$*, the following probability is negligible in* λ*:*

$$\Pr\left[\mu' \neq \mu \;:\; \begin{array}{c} (\mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}) \leftarrow\!\$\; \mathsf{Setup}(1^\lambda, 1^n, 1^t, 1^z) \\ \mathsf{sk}^i_{C_i} \leftarrow\!\$\; \mathsf{KeyGen}(\mathsf{msk}, i, C_i)\ \forall\ i \in \mathcal{S} \\ \mathsf{ct}_x \leftarrow\!\$\; \mathsf{Enc}(\mathsf{mpk}, \mu, x) \\ d_i := \mathsf{Dec}(\mathsf{sk}^i_{C_i}, \mathsf{ct}_x)\ \forall\ i \in \mathcal{S} \\ \mu' := \mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}_x, \{d_j\}_{j\in\mathcal{S}}) \end{array}\right]$$

Semantic Security. We define a semantic security for an ABTE scheme to generalize the standard ABE security and the standard semantic security for threshold encryption. Roughly speaking, we require that an adversary cannot learn anything about the encrypted message for an attribute, even if it is allowed to (i) query arbitrarily many keys for policies not satisfied by the attribute (as in ABE security), (ii) query up to $t-1$ keys for policies satisfied by the attribute (as in threshold encryption security). We also allow the adversary to query decryption shares for any *valid* ciphertext. This definition is stronger than both ABE security and threshold encryption security – if we set the threshold $t = 1$, we get ABE security; on the other hand, if we restrict the policy for all keys to be the accepting function (which is satisfied by all attributes), then we get standard threshold encryption security.

Selective adversaries. Achieving fully adaptive security, wherein the adversary can (i) choose the challenge attribute at any point in the game, and (ii) corrupt parties with satisfying policies at any point in the game, is known to be a non-trivial task, already for attribute-based encryption [21, 65] and for standard threshold encryption [31]. Since the problem of fully-adaptive adversaries is not at the focus of this work, we consider selective adversaries. Specifically, the game starts with the adversary specifying the challenge attribute x^*. The adversary is then given access to a secret-key oracle, which it can use to query:

– Keys for party i and satisfying policy C_i i.e. $C_i(x^*) = 1$. The adversary can only make up to $t-1$ such queries. Moreover, it must make all such queries before making any query for a non-satisfying policy or a partial decryption. We can formalize this by maintaining a bit b

which is set to 1 if the adversary queries the partial decryption oracle or the key oracle for a non-satisfying policy. The secret-key oracle, when given a satisfying policy as input, checks whether this bit has been set, and if so, it ignores the query, returning $\perp$.
- Keys for party i and non-satisfying policy C_i i.e. $C_i(x^*) = 0$. The adversary can query arbitrarily many such keys. Note that, we allow the adversary to query the secret key for any party with respect to arbitrarily many non-satisfying policies.

Moreover, the adversary can query decryption shares for a valid ciphertext for any party and policy. Formally, it provides a valid ciphertext by specifying a message m, an attribute x and randomness r, along with a party i and a policy C_i. The challenger then produces the ciphertext by encrypting μ with respect to x using r, and outputs the decryption share using the key for (i, C_i). We formally define this notion in Definition 10 and Figure 5.

Definition 10 (Selective CPA Security). *We say that* ABTE *satisfies selective CPA security if for every probabilistic polynomial time adversary $\mathcal{A}$, the following function is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{sel-cpa}}_{\mathcal{A},\mathsf{ABTE}}(\lambda) := \left| \frac{1}{2} - \Pr[\mathbf{SelectiveIND\text{-}CPA}_{\mathcal{A},\mathsf{ABTE}}(\lambda) = 1] \right|.$$

Experiment $\mathbf{SelectiveIND\text{-}CPA}_{\mathcal{A},\mathsf{ABTE}}(\lambda)$

$\mathcal{J} := \emptyset$; $(n, t, z, x^*, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$

$(\mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, 1^n, 1^t, 1^z)$

$(\mu_0, \mu_1, \mathsf{state}) \leftarrow\!\$\ \mathcal{A}^{\mathsf{sk}(\cdot),\mathsf{dec}(\cdot)}(\mathsf{state}, \mathsf{mpk}, \mathsf{pkc})$

$b \leftarrow\!\$\ \{0,1\},\ \mathsf{ct}^* \leftarrow\!\$\ \mathsf{Enc}(\mathsf{mpk}, \mu_b, x^*)$

$b' \leftarrow\!\$\ \mathcal{A}^{\mathsf{sk}(\cdot),\mathsf{dec}(\cdot)}(\mathsf{state}, \mathsf{ct}^*)$

if $b' = b$ **then return** 1 **else return** 0

Oracle $\mathsf{sk}(i, C_i)$

if $i \notin [n]$: **return** $\perp$

if $C_i(x^*) = 0$:

  return $\mathsf{sk}^i_{C_i} \leftarrow\!\$\ \mathsf{KeyGen}(\mathsf{msk}, i, C_i)$

if $(i \in \mathcal{J} \wedge C_i \neq \mathcal{J}[i]) \vee (i \notin \mathcal{J} \wedge |\mathcal{J}| = t - 1)$:

  return $\perp$

$\mathcal{J}[i] \leftarrow C_i$

return $\mathsf{sk}^i_{C_i} \leftarrow\!\$\ \mathsf{KeyGen}(\mathsf{msk}, i, C_i)$

Oracle $\mathsf{dec}(\mu, r, x, i, C_i)$

if $i \notin [n] \vee C_i(x) = 0$:

  return $\perp$

$\mathsf{ct} := \mathsf{Enc}(\mathsf{mpk}, \mu, x; r)$

$\mathsf{sk}^i_{C_i} \leftarrow\!\$\ \mathsf{KeyGen}(\mathsf{msk}, i, C_i)$

return $\mathsf{Dec}(\mathsf{sk}^i_{C_i}, \mathsf{ct})$

Fig. 5. The selective IND-CPA security experiment for an attribute-based threshold encryption scheme ABTE and an adversary $\mathcal{A}$

5 Attribute-based Threshold Encryption from Pairings

In this section, we present our first ABTE scheme for the policy class represented by arithmetic span programs. We start with formally defining this class.

5.1 Preliminaries

Definition 11 (Arithmetic Span Programs [53]). *An arithmetic span program (V, ρ) is a collection of vectors $V = \{(\boldsymbol{y}_j, \boldsymbol{z}_j)\}_{j\in[m]}$ where $\boldsymbol{y}_j, \boldsymbol{z}_j \in \mathbb{F}_q^\ell$ and $\rho : [m] \to [z]$. We say that,*

$$x \in \mathbb{F}_q^z \text{ satisfies } (V,\rho) \text{ iff } \boldsymbol{e}_\ell \in \text{ span}\langle x_{\rho(j)}\boldsymbol{y}_j + \boldsymbol{z}_j\rangle,$$

where $\boldsymbol{e}_\ell := (0, 0, \ldots, 0, 1) \in \mathbb{F}_q^\ell$ and span refers to linear span of a collection of column vectors.

5.2 Our Construction

Figure 6 presents our pairings-based ABTE scheme. We use $\mathsf{GroupGen}(1^\lambda)$ to denote a bilinear group generator that outputs an asymmetric bilinear group.

$\underline{\mathsf{Setup}(1^\lambda, 1^n, 1^t, 1^z)}$:

1. Sample a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda)$, sample $\alpha \leftarrow\!\$\ \mathbb{Z}_q$, $a \leftarrow\!\$\ \mathbb{Z}_q^*$ and $\boldsymbol{v} \leftarrow\!\$\ \mathbb{Z}_q^z$.
2. Sample Shamir t-out-of-n sharing $(\alpha_1, \ldots, \alpha_n)$ of α.
3. Output $\mathsf{mpk} \leftarrow (e(g_1, g_2)^\alpha, g_1, g_2, g_1^a, g_1^{\boldsymbol{v}})$ and $\mathsf{msk} \leftarrow (\{\alpha_i\}_{i\in[n]}, a, \boldsymbol{v})$.

$\underline{\mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, i, (V, \rho))}$:

1. Sample $\boldsymbol{u} \leftarrow\!\$\ \mathbb{Z}_q^\ell$ subject to the constraint $\langle \boldsymbol{u}, \boldsymbol{e}_\ell\rangle = \alpha_i$.
2. Compute $\beta_j := \langle \boldsymbol{u}, \boldsymbol{y}_j\rangle$, $\gamma_j := \langle \boldsymbol{u}, \boldsymbol{z}_j\rangle$ for all $j \in [m]$.
3. For all $j \in [m]$, compute $k_{j,0} := g_2^{\frac{\beta_j}{a}}$ and $k_{j,1} := g_2^{\gamma_j - \frac{\beta_j \boldsymbol{v}_{\rho(j)}}{a}}$.
4. Output $\mathsf{sk}^i_{V,\rho} \leftarrow \{k_{j,0}, k_{j,1}\}_{j\in[m]} \in \mathbb{G}_2^{2m}$.

$\underline{\mathsf{Enc}(\mathsf{mpk}, x, \mu)}$: // $x \in \mathbb{F}_q^z$

1. Sample $s \leftarrow\!\$\ \mathbb{Z}_q$.
2. Compute $C_i := g_1^{(ax_i + \boldsymbol{v}_i)s}\ \forall\ i \in [z]$.
3. Output $\mathsf{ct}_x := (C_0 := g_1^s,\ \{C_i\}_{i\in[z]},\ C' := e(g_1, g_2)^{\alpha s} \cdot \mu)$

$\underline{\mathsf{Dec}(\mathsf{sk}^i_{V,\rho}, \mathsf{ct}_x)}$:

1. If $\boldsymbol{e}_\ell \notin \text{span}\langle x_{\rho(j)}\boldsymbol{y}_j + \boldsymbol{z}_j\rangle$, output $\perp$.
2. Otherwise, compute $(w_1, w_2, \ldots, w_m) \in \mathbb{Z}_q^m$ such that $\boldsymbol{e}_\ell = \sum_{j\in[m]} w_j(x_{\rho(j)}\boldsymbol{y}_j + \boldsymbol{z}_j)$.
3. Parse ct_x as $(C_0, \{C_i\}_{i\in[z]},\ C')$ and $\mathsf{sk}^i_{V,\rho}$ as $\{k_{j,0}, k_{j,1}\}_{j\in[m]}$.
4. Output $d_i := \prod_{j=1}^{m} \left(e(C_0, k_{j,1}) \cdot e(C_{\rho(j)}, k_{j,0})\right)^{w_j}$.

$\underline{\mathsf{Combine}(\mathsf{ct}_x, \{d_i\}_{i\in\mathcal{S}})}$:

1. Parse ct_x as $(C_0, \{C_i\}_{i\in[z]},\ C')$.
2. Output $C' \cdot \prod_{i\in\mathcal{S}} d_i^{-\lambda_{i,\mathcal{S},0}}$, where $\lambda_{i,\mathcal{S},u} := \prod_{j\in\mathcal{S}\setminus\{i\}} \left(\frac{j-u}{j-i}\right)$ for all $i \in \mathcal{S}$.

Fig. 6. Our Attribute-based Threshold Encryption scheme from pairings P-ABTE.

Correctness. Theorem 4 below proves correctness of our pairings-based ABTE construction.

Theorem 4. *The pairings-based attribute-based threshold encryption scheme* P-ABTE *has zero correctness error.*

Proof (of Theorem 4). Let us consider a ciphertext ct_x encrypting a message μ and attribute x, and any set $\mathcal{S}$ of size $\geq t$ such that all parties in $\mathcal{S}$ have a satisfying policy with respect to x. Formally, let $V^{(i)}, \rho^{(i)}$ denote the policy for party i, where $V^{(i)} = \{\boldsymbol{y}_j^{(i)}, \boldsymbol{z}_j^{(i)}\}_{j\in[m]}$. We have that, for all $i \in \mathcal{S}$, $\boldsymbol{e}_\ell \in \text{span}\langle x_{\rho^{(i)}(j)}\boldsymbol{y}_j + \boldsymbol{z}_j\rangle$. This means, that Dec when run with $\mathsf{sk}^i_{V^{(i)},\rho^{(i)}}, \mathsf{ct}_x$ will not output $\perp$, for all $i \in \mathcal{S}$. Next, let us use $\boldsymbol{u}^{(i)}, \beta_j^{(i)}, \gamma_j^{(i)}$ to denote the parameters used during key generation for party i, and let $w_1^{(i)}, \ldots, w_m^{(i)}$ denote the coefficients computing during the execution of Dec with the key of party i and ct_x. We will now expand on the expression of d_i:

$$
\begin{aligned}
d_i &= \prod_{j=1}^{m} \left(e\left(g_1^s, g_2^{\gamma_j^{(i)} - \frac{\beta_j^{(i)} \boldsymbol{v}_{\rho^{(i)}(j)}}{a}} \right) \cdot e\left(g_1^{\left(a x_{\rho^{(i)}(j)} + \boldsymbol{v}_{\rho^{(i)}(j)}\right) s}, g_2^{\frac{\beta_j^{(i)}}{a}} \right) \right)^{w_j^{(i)}} \\
&= \prod_{j=1}^{m} \left(e(g_1, g_2)^{s\gamma_j^{(i)} + s x_{\rho^{(i)}(j)} \beta_j^{(i)}} \right)^{w_j^{(i)}} \\
&= \prod_{j=1}^{m} \left(e(g_1, g_2)^{s < \boldsymbol{u}^{(i)}, (x_{\rho^{(i)}(j)} \boldsymbol{y}_j^{(i)} + \boldsymbol{z}_j^{(i)}) >} \right)^{w_j^{(i)}} \\
&= e(g_1, g_2)^{s < \boldsymbol{u}^{(i)}, \Sigma_{j\in[m]} (w_j^{(i)} \cdot (x_{\rho^{(i)}(j)} \boldsymbol{y}_j^{(i)} + \boldsymbol{z}_j^{(i)})) >} \\
&= e(g_1, g_2)^{s\langle \boldsymbol{u}^{(i)}, \boldsymbol{e}_\ell \rangle} \qquad (1) \\
&= e(g_1, g_2)^{s\alpha_i}
\end{aligned}
$$

Equation 1 follows from the fact that the coefficients $\{w_j^{(i)}\}_{j\in[m]}$ are computed such that $\boldsymbol{e}_\ell = \Sigma_{j\in[m]} w_j^{(i)} (x_{\rho^{(i)}(j)} \boldsymbol{y}_j^{(i)} + \boldsymbol{z}_j^{(i)})$.

Lastly, we observe that the $\mathsf{Combine}$ algorithm simply does Lagrange interpolation in the exponent. More formally, the output of $\mathsf{Combine}$ is:

$$
\begin{aligned}
C' \cdot \prod_{i\in\mathcal{S}} d_i^{-\lambda_{i,\mathcal{S},0}} &= C' \cdot \prod_{i\in\mathcal{S}} (e(g_1, g_2)^{s\alpha_i})^{-\lambda_{i,\mathcal{S},0}} \\
&= \mu \cdot e(g_1, g_2)^{s\alpha} \cdot e(g_1, g_2)^{-s\cdot(\Sigma_{i\in[m]} \lambda_{i,\mathcal{S},0} \cdot \alpha_i)} \\
&= \mu
\end{aligned}
$$

□

Security. We prove security of our ABTE scheme $\mathsf{P\text{-}ABTE}$ by relying on the decisional bilinear Diffie-Hellman assumption for asymmetric groups, stated in Definition 12 below. Theorem 5 below formally proves this.

Definition 12. *Let $\mathcal{G}$ be an asymmetric bilinear group generator. The decisional bilinear Diffie-Hellman assumption holds with respect to $\mathcal{G}$ if, for all* PPT *adversaries $\mathcal{A}$, the following function is*

negligible in λ:

$$\mathsf{Adv}^{\mathrm{dbdh}}_{\mathcal{G},\mathcal{A}}(\lambda) := \left| \Pr\left[b' = b^* : \begin{array}{c} (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e) \leftarrow\$ \mathcal{G}(1^\lambda) \\ a, b, s, z \leftarrow\$ \mathbb{Z}_p \\ h_0 := e(g_1, g_2)^{abs}, h_1 := e(g_1, g_2)^z \\ b^* \leftarrow\$ \{0,1\} \\ b' \leftarrow\$ \mathcal{A}\left(\begin{array}{c} \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e, \\ g_1^a, g_2^a, g_2^b, g_1^s, h_{b^*} \end{array} \right) \end{array} \right] - \frac{1}{2} \right|.$$

Theorem 5. *For all* PPT *adversaries $\mathcal{A}$, there exists another* PPT *adversary $\mathcal{B}$ such that,*

$$\mathsf{Adv}^{\mathrm{sel-cpa}}_{\mathcal{A},\mathsf{P\text{-}ABTE}}(\lambda) \leq 2\left(\mathsf{Adv}^{\mathrm{dbdh}}_{\mathsf{GroupGen},\mathcal{B}}(\lambda) + \frac{1}{q} \right),$$

where q is the size of the group output by the group generator GroupGen.

Proof. Consider any adversary $\mathcal{A}$ playing the CPA game. Recall that in this game, the adversary is allowed to query up to $t-1$ keys for policies that are satisfied by the challenge attribute, along with an arbitrary number of key queries for non-satisfying policies, as well as partial decryptions for valid ciphertexts. Note that we are considering a selective game, wherein the adversary makes all the secret-key queries for satisfying policies before making any other query. We will construct an adversary $\mathcal{B}$ that will act as the challenger to $\mathcal{A}$ and use it to break the decisional bilinear Diffie-Hellman (DBDH) assumption. Specifically, $\mathcal{B}$ does the following:

- Receive a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$, group elements $A_1 = g_1^a, A_2 = g_2^a, B_2 = g_2^b, S_1 = g_1^s$ and a target group element h from the DBDH challenger, that could be either $e(g_1, g_2)^{abs}$ or a uniformly random element from $\mathbb{G}_T$.
- Abort if $a = 0$ i.e. if $A_1 = g_1^0$. Otherwise, receive n, t, z and the challenge attribute x^* from $\mathcal{A}$.
- Sample $\tilde{\boldsymbol{v}} \leftarrow\$ \mathbb{Z}_q^z$ and implicitly set $\alpha = ab$ and $\boldsymbol{v} = \tilde{\boldsymbol{v}} - a\boldsymbol{x}^*$. Note that $\mathcal{B}$ does not know the value of α, but we will show it can simulate the game regardless.
- Send $\mathsf{mpk} = (e(A_1, B_2), g_1, g_2, A_1, g^{\tilde{\boldsymbol{v}}}/A_1^{x^*})$ to $\mathcal{A}$. It is easy to see that this is identically distributed as the real public key. Initialize $\mathcal{J} := \emptyset$, and a map $M : [n] \to \mathbb{Z}_q$ that initially empty.
- On receiving a satisfying key query $(i, (V, \rho))$, if $i \notin [n]$ or if $i \notin \mathcal{J}$ and $|\mathcal{J}| = t - 1$, abort. Otherwise, if $i \in \mathcal{J}$, retrieve $\hat{\alpha}_i \leftarrow M[i]$. If $i \notin \mathcal{J}$, set $\mathcal{J} \leftarrow \mathcal{J} \cup \{i\}$ and sample $\hat{\alpha}_i \leftarrow\$ \mathbb{Z}_q$ and implicitly set α_i i.e. ith Shamir share of α to be $a\hat{\alpha}_i$. Also set $M[i] = \hat{\alpha}_i$.
  Then, sample $\boldsymbol{u}^* \leftarrow\$ \mathbb{Z}_q^\ell$ subject to $\langle \boldsymbol{u}^*, \boldsymbol{e}_\ell \rangle = \hat{\alpha}_i$, and implicitly set $\boldsymbol{u} = a\boldsymbol{u}^*$. For all $j \in [m]$, compute $k_{j,0} = g_2^{\langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle}$ and $k_{j,1} = A_2^{\langle \boldsymbol{u}^*, \boldsymbol{z}_j \rangle} \cdot (g_2^{\boldsymbol{v}_{\rho(j)}})^{-\langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle}$, where $g_2^{\boldsymbol{v}} = g_2^{\tilde{\boldsymbol{v}}}/A_2^{x^*}$. To see why these keys are distributed correctly, observe that,

$$\frac{\beta_j}{a} = \langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle \ ; \ \gamma_j - \frac{\beta_j \boldsymbol{v}_{\rho(j)}}{a} = a\langle \boldsymbol{u}^*, \boldsymbol{z}_j \rangle - \langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle \boldsymbol{v}_{\rho(j)}.$$

  $\mathcal{B}$ simply returns $\{k_{j,0}, k_{j,1}\}_{j \in [m]}$ to $\mathcal{A}$.
- On receiving a non-satisfying key query $(i, (V, \rho))$, $\mathcal{B}$ computes $\boldsymbol{u}^* \in \mathbb{Z}_q^\ell$ such that

$$\langle \boldsymbol{u}^*, \boldsymbol{e}_\ell \rangle = 1 \text{ and } \langle \boldsymbol{u}^*, x^*_{\rho(j)} \boldsymbol{y}_j + \boldsymbol{z}_j \rangle = 0 \ \forall \ j \in [m].$$

  Note that such a $\boldsymbol{u}^*$ must exist given that x^* does not satisfy (V, ρ). Next, sample $\tilde{\boldsymbol{u}} \leftarrow\$ \mathbb{Z}_q^\ell$ such that $\langle \tilde{\boldsymbol{u}}, \boldsymbol{e}_\ell \rangle = 0$ (Note that since we are considering the selective game, $\mathcal{A}$ has already made all the satisfying key queries by this point). Next,

1. If $i \in \mathcal{J}$, retrieve $\hat{\alpha}_i = M[i]$ and implicitly set $\boldsymbol{u} = a\hat{\alpha}_i \boldsymbol{u}^* + a\tilde{\boldsymbol{u}}$. This is distributed identically to an honestly sampled $\boldsymbol{u}$ because:

$$\langle \boldsymbol{u}, \boldsymbol{e}_\ell \rangle = a\hat{\alpha}_i \langle \boldsymbol{u}^*, \boldsymbol{e}_\ell \rangle + a\langle \tilde{\boldsymbol{u}}, \boldsymbol{e}_\ell \rangle = a\hat{\alpha}_i = \alpha_i.$$

The last equality follows from the fact that we implicitly set α_i to be $a\hat{\alpha}_i$.
Compute $k_{j,0} = g_2^{\langle \hat{\alpha}_i \boldsymbol{u}^* + \tilde{\boldsymbol{u}}, \boldsymbol{y}_j \rangle}$ and $k_{j,1} = k_{j,0}^{-\tilde{\boldsymbol{v}}_{\rho(j)}} A_2^{\langle \tilde{\boldsymbol{u}}, \boldsymbol{z}_j + x^*_{\rho(j)} \boldsymbol{y}_j \rangle}$. To see why this is correctly distributed, observe that:

$$\frac{\beta_j}{a} = \langle \frac{\boldsymbol{u}}{a}, \boldsymbol{y}_j \rangle = \hat{\alpha}_i \langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle + \langle \tilde{\boldsymbol{u}}, \boldsymbol{y}_j \rangle$$

Moreover, using the fact that $\langle \boldsymbol{u}^*, \boldsymbol{z}_j + x^*_{\rho(j)} \boldsymbol{y}_j \rangle = 0$, we get:

$$\begin{aligned}
\gamma_j - \frac{\beta_j \boldsymbol{v}_{\rho(j)}}{a} &= \gamma_j - \frac{\beta_j \tilde{\boldsymbol{v}}_{\rho(j)}}{a} + \beta_j x^*_{\rho(j)} \\
&= -\frac{\beta_j \tilde{\boldsymbol{v}}_{\rho(j)}}{a} + \langle \boldsymbol{u}, \boldsymbol{z}_j + x^*_{\rho(j)} \boldsymbol{y}_j \rangle \\
&= -\frac{\beta_j \tilde{\boldsymbol{v}}_{\rho(j)}}{a} + a\hat{\alpha}_i \langle \boldsymbol{u}^*, \boldsymbol{z}_j + x^*_{\rho(j)} \boldsymbol{y}_j \rangle + a\langle \tilde{\boldsymbol{u}}, \boldsymbol{z}_j + x^*_{\rho(j)} \boldsymbol{y}_j \rangle \\
&= -\frac{\beta_j \tilde{\boldsymbol{v}}_{\rho(j)}}{a} + a\langle \tilde{\boldsymbol{u}}, \boldsymbol{z}_j + x^*_{\rho(j)} \boldsymbol{y}_j \rangle
\end{aligned}$$

Hence, the keys prepared by $\mathcal{B}$ are indeed correctly distributed. $\mathcal{B}$ returns $\{k_{j,0}, k_{j,1}\}$ to $\mathcal{A}$.
2. If $i \notin \mathcal{J}$ and $|\mathcal{J}| < t-1$, then sample $\hat{\alpha}_i \leftarrow\$ \mathbb{Z}_q$, set $M[i] = \hat{\alpha}_i$, and $\mathcal{J} \leftarrow \mathcal{J} \cup \{i\}$. Then, respond to the query exactly as in Case 1 above.
3. If $i \notin \mathcal{J}$ and $|\mathcal{J}| = t-1$, let $\mathcal{S} := \mathcal{J} \cup \{0\}$, then, implicitly set $\alpha_i = \lambda_{0,\mathcal{S},i} \cdot ab + \Sigma_{j \in \mathcal{J}} \lambda_{j,\mathcal{S},i} \cdot a\hat{\alpha}_j$. Essentially, we are defining a polynomial f of degree $t-1$ which evaluates to ab at zero, and $f(i) = \alpha_i = a\hat{\alpha}_i$ for all $i \in \mathcal{J}$. This is the polynomial that we have implicitly used to generate Shamir shares of α. Let us write $\alpha_i = a\hat{\alpha}_i$, where $\hat{\alpha}_i$ is implicitly set to $\lambda_{0,\mathcal{S},i} \cdot b + \Sigma_{j \in \mathcal{J}} \lambda_{j,\mathcal{S},i} \cdot \hat{\alpha}_j$. Next, $\mathcal{B}$ implicitly sets $\boldsymbol{u} = \alpha_i \boldsymbol{u}^* + a\tilde{\boldsymbol{u}}$. By the same argument as above, this is distributed exactly as a real key. For each $j \in [m]$, $\mathcal{B}$ computes

$$k_{j,0} = B_2^{\lambda_{0,\mathcal{S},i} \langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle} \cdot g_2^{\langle \tilde{\boldsymbol{u}}, \boldsymbol{y}_j \rangle + (\Sigma_{j \in \mathcal{J}} \lambda_{j,\mathcal{S},i} \cdot \hat{\alpha}_j) \langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle}.$$

To see why this is correct, observe that,

$$\frac{\beta_j}{a} = \hat{\alpha}_i \langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle + \langle \tilde{\boldsymbol{u}}, \boldsymbol{y}_j \rangle = (\lambda_{0,\mathcal{S},i} \langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle) b + (\Sigma_{j \in \mathcal{J}} \lambda_{j,\mathcal{S},i} \cdot \hat{\alpha}_j) \langle \boldsymbol{u}^*, \boldsymbol{y}_j \rangle + \langle \tilde{\boldsymbol{u}}, \boldsymbol{y}_j \rangle$$

Lastly, for each $j \in [m]$, $\mathcal{B}$ computes $k_{j,1} = k_{j,0}^{-\tilde{\boldsymbol{v}}_{\rho(j)}} \cdot A_2^{\langle \tilde{\boldsymbol{u}}, \boldsymbol{z}_j + x^*_{\rho(j)} \boldsymbol{y}_j \rangle}$. By a calculation similar to case 1 above, this key is indeed correct. $\mathcal{B}$ then responds to $\mathcal{A}$ with $\{k_{j,0}, k_{j,1}\}_{j \in [m]}$.

– On receiving a partial decryption query for message μ, randomness r, attribute x, and party i with policy C_i, $\mathcal{B}$ simply aborts if $i \notin [n]$ or if $C_i(x) = 0$. This is similar to an honest challenger for the CPA game. Otherwise, if $i \in \mathcal{J}$, then $\mathcal{B}$ responds with $e(A_1, g_2)^{r\hat{\alpha}_i}$. To see why this is correct, it can be shown that $d_i = e(g_1, g_2)^{r\alpha_i}$ (see proof of Thm. 4), where r is the randomness used to generate the ciphertext. Moreover, $\alpha_i = a\hat{\alpha}_i$ for all $i \in \mathcal{J}$. Hence, $e(g_1, g_2)^{r\alpha_i} = e(g_1, g_2)^{ra\hat{\alpha}_i} = e(A_1, g_2)^{r\hat{\alpha}_i}$, as desired.

If $i \notin \mathcal{J}$, but $|\mathcal{J}| < t-1$, then $\mathcal{B}$ samples $\hat{\alpha}_i \leftarrow\$ \mathbb{Z}_q$, sets $M[i] = \hat{\alpha}_i$ and $\mathcal{J} \leftarrow \mathcal{J} \cup \{i\}$. It then answers the query in the same way as above, i.e. $e(A_1, g_2)^{r\hat{\alpha}_i}$.
Lastly, if $i \notin \mathcal{J}$, but $|\mathcal{J}| = t-1$, let $\mathcal{S} = \mathcal{J} \cup \{0\}$. $\mathcal{B}$ responds with $e(A_1, B_2)^{r \cdot \lambda_{0,\mathcal{S},i}} \cdot e(A_1, g_2)^{r(\Sigma_{j \in \mathcal{J}} \lambda_{j,\mathcal{S},i} \cdot \hat{\alpha}_j)}$. Correctness follows directly from the fact that $\alpha_i = \lambda_{0,\mathcal{S},i} \cdot ab + \Sigma_{j \in \mathcal{J}} \lambda_{j,\mathcal{S},i} \cdot a\hat{\alpha}_j$, as seen in case 3 above.

Eventually, $\mathcal{B}$ receives messages μ_0, μ_1 from $\mathcal{A}$. $\mathcal{B}$ picks $\beta^* \leftarrow\$ \{0,1\}$, and computes the challenge ciphertext: $\mathsf{ct}^* = (S_1, S_1^{\tilde{\boldsymbol{v}}}, \mu_{\beta^*} \cdot h)$. Let β' denote $\mathcal{A}$'s final output. $\mathcal{B}$ outputs 0 if $\beta' = \beta^*$ and 1 otherwise.

Let us now analyse the advantage of $\mathcal{B}$. Let E denote the event that $\mathcal{B}$ aborts because $A = g$. We claim that in the event $\neg\mathsf{E}$, $\mathcal{B}$ perfectly simulates the real challenger to $\mathcal{A}$. As we have already argued above, the mpk, the secret key responses and the partial decryption responses are distributed identically to the real ones. For the ciphertext, observe that C_0 is identically distributed, as implicitly, $\mathcal{B}$ has used s as the randomness for this ciphertext. Next, for C_i, recall that $ax_i^* + \boldsymbol{v}_i = ax_i^* + \tilde{\boldsymbol{v}}_i - ax_i^* = \tilde{\boldsymbol{v}}_i$, hence, these terms are also correct. Lastly, if h equals $e(g,g)^{abs}$, then ct^* is indeed a valid encryption of μ_{β^*}. On the other hand, if h is a random element in $\mathbb{G}_T$, then ct^* is the encryption of a random message, independent of β^*. Hence, $\mathcal{B}$ perfectly simulates the CPA game.

Note that the event E only occurs with probability $1/q$. Then, by a standard analysis, we get that,

$$\mathsf{Adv}^{\mathrm{dbdh}}_{\mathsf{GroupGen},\mathcal{B}}(\lambda) = \frac{1}{2}\left(1 - \frac{1}{q}\right)\mathsf{Adv}^{\mathrm{combined-cpa}}_{\mathcal{A},\text{P-ABTE}}(\lambda) \geq \frac{1}{2}\mathsf{Adv}^{\mathrm{combined-cpa}}_{\mathcal{A},\text{P-ABTE}}(\lambda) - \frac{1}{q}.$$

□

6 Attribute-based Threshold Encryption from Lattices

In this section, we present our attribute-based ramp-threshold encryption scheme for the policies represented by arithmetic circuits. We start with some preliminaries in Section 6.1 and 6.2, and describe our construction in Section 6.3.

6.1 Preliminaries on Lattices

Lattices. Let q, ℓ, m be positive integers. For a matrix $\boldsymbol{A} \in \mathbb{Z}_q^{\ell \times m}$ we let $\Lambda_q^{\perp}(\boldsymbol{A})$ denote the lattice $\{\boldsymbol{x} \in \mathbb{Z}^m : \boldsymbol{A}\boldsymbol{x} = 0 \text{ in } \mathbb{Z}_q\}$. More generally, for $\boldsymbol{u} \in \mathbb{Z}_q^{\ell}$ we let $\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A})$ denote the coset $\{\boldsymbol{x} \in \mathbb{Z}^m : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{u} \text{ in } \mathbb{Z}_q\}$.

Definition 13 (Learning with errors (LWE) [60]). *Fix integers ℓ, m, a prime integer q and a noise distribution χ over $\mathbb{Z}$. The (ℓ, m, q, χ)-LWE problem is to distinguish the following two distributions:*

$$(\boldsymbol{A}, \boldsymbol{A}^{\top}\boldsymbol{s} + \boldsymbol{e}) \quad \textit{and} \quad (\boldsymbol{A}, \boldsymbol{u})$$

where $\boldsymbol{A} \leftarrow\$ \mathbb{Z}_q^{\ell \times m}$, $\boldsymbol{s} \leftarrow\$ \mathbb{Z}_q^{\ell}$, $\boldsymbol{e} \leftarrow\$ \chi^m$, $\boldsymbol{u} \leftarrow\$ \mathbb{Z}_q^m$ are independently sampled.

Throughout the paper we always set $m = \Theta(\ell \log q)$ and simply refer to the (ℓ, q, χ)-LWE problem.

Matrix norms. For a vector $\boldsymbol{u}$ we let $\|\boldsymbol{u}\|$ denote its ℓ_2 norm. We use $\|\boldsymbol{R}\|$ to denote the ℓ_2 length of the longest column of $\boldsymbol{R}$. $\|\boldsymbol{R}\|_2$ is the operator norm of $\boldsymbol{R}$ defined as $\|\boldsymbol{R}\|_2 = \sup_{\|\boldsymbol{x}\|=1} \|\boldsymbol{R}\boldsymbol{x}\|$.

Discrete Gaussians. Regev [60] defined a natural distribution on $\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A})$ called a discrete Gaussian parameterized by a scalar $\sigma > 0$. We use $\mathcal{D}_{\sigma}(\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}))$ to denote this distribution. For a random

matrix $\boldsymbol{A} \in \mathbb{Z}_q^{\ell \times m}$ and $\sigma = \tilde{\Omega}(\sqrt{\ell})$, a vector $\boldsymbol{x}$ sampled from $\mathcal{D}_\sigma(\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}))$ has ℓ_2 norm less than $\sigma\sqrt{m}$ with probability at least $1 - \mathrm{negl}(m)$. For a matrix $\boldsymbol{U} = (\boldsymbol{u}_1 | \cdots | \boldsymbol{u}_k) \in \mathbb{Z}_q^{\ell \times k}$, we let $\mathcal{D}_\sigma(\Lambda_q^{\boldsymbol{U}}(\boldsymbol{A}))$ be a distribution on matrices in $\mathbb{Z}^{m \times k}$ where the i-th column is sampled from $\mathcal{D}_\sigma(\Lambda_q^{\boldsymbol{u}_i}(\boldsymbol{A}))$ independently for $i = 1, \ldots, k$.

Lemma 1 (Trapdoor generators [13]). *Let $\ell, m, q > 0$ be integers with q prime. There are polynomial time algorithms with the properties below:*

- $\mathsf{TrapGen}(1^\ell, 1^m, q) \to (\boldsymbol{A}, \boldsymbol{T_A})$ *([3, 5, 55]): a randomized algorithm that, when $m = \Theta(\ell \log q)$, outputs a full-rank matrix $\boldsymbol{A} \in \mathbb{Z}_q^{\ell \times m}$ and basis $\boldsymbol{T_A} \in \mathbb{Z}^{m \times m}$ for $\Lambda_q^\perp(\boldsymbol{A})$ such that $\boldsymbol{A}$ is negl(ℓ)-close to uniform and $\|\boldsymbol{T_A}\|_{\mathsf{GS}} = O(\sqrt{\ell \log q})$, with all but negligible probability in ℓ.*
- $\mathsf{SampleD}(\boldsymbol{A}, \boldsymbol{T_A}, \boldsymbol{U}, \sigma)$ *([GPV08]): a randomized algorithm that, given matrix $\boldsymbol{A} \in \mathbb{Z}_q^{\ell \times m}$, matrix $\boldsymbol{U} \in \mathbb{Z}_q^{\ell \times k}$, a basis $\boldsymbol{T_A}$ of $\Lambda_q^\perp(\boldsymbol{A})$ and $\sigma = \|\boldsymbol{T_A}\|_{GS} \cdot \omega(\sqrt{\log m})$, outputs a random sample $\boldsymbol{X}$ from a distribution that is statistically close to $\mathcal{D}_\sigma(\Lambda_q^{\boldsymbol{U}}(\boldsymbol{A}))$.*
- *For $m = \ell \lceil \log q \rceil$ there is a fixed full-rank matrix $\boldsymbol{G} \in \mathbb{Z}_q^{\ell \times m}$ s.t. the lattice $\Lambda_q^\perp(\boldsymbol{G})$ has a publicly known short basis $\boldsymbol{T_G} \in \mathbb{Z}^{m \times m}$. We use $\boldsymbol{G}^{-1}(\boldsymbol{A})$ to denote the binary decomposition of any matrix $\boldsymbol{A} \in \mathbb{Z}_q^{\ell \times m}$, such that (i) $\boldsymbol{G} \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}) = \boldsymbol{A}$ and (ii) $\|\boldsymbol{R}\|_2 \leq m$ and $\|\boldsymbol{R}^\top\|_2 \leq m$.*
- $\mathsf{SampleRight}(\boldsymbol{A}, \boldsymbol{T_A}, \boldsymbol{B}, \boldsymbol{U}, \sigma)$ *: a randomized algorithm that given full-rank matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{Z}_q^{\ell \times m}$, matrix $\boldsymbol{U} \in \mathbb{Z}_q^{\ell \times m}$, a basis $\boldsymbol{T_A}$ of $\Lambda_q^\perp(\boldsymbol{A})$ and $\sigma = \|\boldsymbol{T_A}\|_{GS} \cdot \omega(\sqrt{\log m})$, outputs a random sample $\boldsymbol{X} \in \mathbb{Z}_q^{2m \times m}$ from a distribution that is statistically close to $\mathcal{D}_\sigma(\Lambda_q^{\boldsymbol{U}}((\boldsymbol{A}|\boldsymbol{B})))$.*

Well-sampledness. Following [46, 32], we require that the aforementioned sampling procedures output well-sampled elements. That is, the preimage outputted by $\mathsf{SampleD}$ with a uniformly random vector/matrix is indistinguishable from a vector/matrix with entries drawn from an appropriate Gaussian distribution. Figure 7 and Definition 14 below formalize this property.

Experiment $\mathbf{Exp}^{\mathrm{preimg},q,\sigma}_{LT,\mathcal{A}}(\lambda)$	Oracle $\mathsf{pre}()$
$(\ell, m, \mathsf{state}) \leftarrow\!\!\$\ \mathcal{A}(1^\lambda)$	$\boldsymbol{u} \leftarrow\!\!\$\ \mathbb{Z}_q^\ell$
$(\boldsymbol{A}, \boldsymbol{T_A}) \leftarrow\!\!\$\ \mathsf{TrapGen}(1^\ell, 1^m, q)$	$\boldsymbol{x}_0 \leftarrow\!\!\$\ \mathsf{SampleD}(\boldsymbol{A}, \boldsymbol{T_A}, \boldsymbol{u}, \sigma)$
$b \leftarrow\!\!\$\ \{0,1\}$	$\boldsymbol{x}_1 \leftarrow\!\!\$\ \mathcal{D}^m_{\mathbb{Z},\sigma}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{pre}()}(\mathsf{state}, \boldsymbol{A})$	**return** $\boldsymbol{x}_b$
return $b' = b$	

Fig. 7. The experiment for well-sampledness of preimage property.

Definition 14 (Well-Sampledness of Preimage). *Fix any function $q : \mathbb{N} \to \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{N}$. The procedure $\mathsf{SampleS}$ is said to satisfy the (q, σ)-well-sampledness property if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathrm{preimg,q},\sigma}_{\mathsf{TrapGen},\mathcal{A}}(\lambda) := \left| \Pr\left[\mathbf{Exp}^{preimg,q,\sigma}_{LT,\mathcal{A}}(\lambda) = 1 \right] - 1/2 \right| \leq negl(\lambda),$$

where $\mathbf{Exp}^{preimg,q,\sigma}_{LT,\mathcal{A}}$ is defined in Fig. 7.

The above property is satisfied by the gadget-based trapdoor lattice sampler presented in [55].

Lemma 2 (Key-Homomorphic Evaluation [13]). *Let $\mathcal{F}_{ckt} = \{\mathcal{F}_{ckt,\lambda}\}_\lambda$ denote the family of polynomial-size arithmetic circuits over $\mathbb{Z}_q$. Let $d = d(\lambda)$ and $k = k(\lambda)$ denote upper bounds on the depth and fan-in of gates across circuits in $\mathcal{F}_{ckt,\lambda}$. Also let $p = p(\lambda)$ denote the upper bound on at least one of the inputs to each multiplication gate in circuits in $\mathcal{F}_{ckt,\lambda}$. Then, there exist polynomial-time deterministic algorithms $\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}, \mathsf{Eval}_{\mathsf{sim}}$ for $\mathcal{F}_{ckt}$ such that:*

- $\mathsf{Eval}_{\mathsf{pk}}(f \in \mathcal{F}_{ckt}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_z \in \mathbb{Z}_q^{\ell \times m}) \to \boldsymbol{B}_f \in \mathbb{Z}_q^{\ell \times m}$.
- $\mathsf{Eval}_{\mathsf{ct}}(f \in \mathcal{F}_{ckt}, ((x_i, \boldsymbol{B}_i, c_i))_{i=1}^z) \to c_f \in \mathbb{Z}_q^m$. *Here $x_i \in \mathbb{Z}_q, \boldsymbol{B}_i \in \mathbb{Z}_q^{\ell \times m}$ and $c_i \in E_{s,\delta}(x_i, \boldsymbol{B}_i)$ for some $\boldsymbol{s} \in \mathbb{Z}_q^\ell$ and $\delta > 0$, where $E_{s,\delta}$ is defined as:*

$$E_{s,\delta}(x, \boldsymbol{B}) = \{(x\boldsymbol{G} + \boldsymbol{B})^\top \boldsymbol{s} + \boldsymbol{e} \in \mathbb{Z}_q^m \text{ where } \|\boldsymbol{e}\| \leq \delta.\}$$

  Note that the same $\mathbf{s}$ is used for all c_i. The output c_f satisfies $c_f \in E_{s,\Delta}(f(x), \boldsymbol{B}_f)$ where $\boldsymbol{B}_f = \mathsf{Eval}_{\mathsf{pk}}(f, (\boldsymbol{B}_1, \ldots, \boldsymbol{B}_z))$. Moreover, $\Delta < \delta \cdot \alpha_{\mathcal{F}}(\ell)$ where $\alpha_{\mathcal{F}}(\ell) = O((p^{k-1}m)^d\sqrt{m})$. This algorithm captures the key-homomorphic property: it translates ciphertexts encrypted under public-keys $\{x_i\}_{i=1}^\ell$ into a ciphertext c_f encrypted under public-key $(f(x), f)$.
- $\mathsf{Eval}_{\mathsf{sim}}(f \in \mathcal{F}_{ckt}, ((x_i^*, \boldsymbol{S}_i))_{i=1}^z, \boldsymbol{A}) \longrightarrow \boldsymbol{S}_f \in \mathbb{Z}_q^{m \times m}$. *Here $x_i^* \in \mathbb{Z}_q$ and $\boldsymbol{S}_i \in \mathbb{Z}_q^{m \times m}$. With $x^* = (x_1^*, \ldots, x_n^*)$, the output $\boldsymbol{S}_f$ satisfies*

$$\boldsymbol{A}\boldsymbol{S}_f - f(\mathbf{x}^*)\boldsymbol{G} = \boldsymbol{B}_f \text{ where } \boldsymbol{B}_f = \mathsf{Eval}_{\mathsf{pk}}(f, (\boldsymbol{A}\boldsymbol{S}_1 - x_1^*\boldsymbol{G}, \ldots, \boldsymbol{A}\boldsymbol{S}_z - x_z^*\boldsymbol{G})).$$

  Lastly, for all $f \in \mathcal{F}_{ckt}$, if $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_z$ are random matrices in $\{\pm 1\}^{m \times m}$ then $\|\boldsymbol{S}_f\|_2 < \alpha_{\mathcal{F}}(\ell)$ with all but negligible probability.

6.2 Preliminaries on Ramp Secret Sharing

We begin by recalling the notion of a partial access structures, that defines authorized and unauthorized sets while allowing a gap between them. A *ramp threshold access structure* is a partial access structures with two thresholds, where all sets smaller than the first threshold are unauthorized and all sets larger than the second threshold are authorized.

Definition 15 (Ramp threshold access structure). *A partial access structure over n parties is a pair $\Gamma = (\Gamma_0, \Gamma_1)$ where $\Gamma_0, \Gamma_1 \subseteq 2^{[n]}$ are non-empty collections of sets such that $B \not\subseteq A$ for every $A \in \Gamma_0, B \in \Gamma_1$. Sets in Γ_1 are called authorized, and sets in Γ_0 are called unauthorized.*

For $0 < \tau_p < \tau_c \leq 1$, the (τ_p, τ_c)-ramp access structure over n parties $\Gamma = (\Gamma_0, \Gamma_1)$ is defined by letting Γ_0 be the collection of all subsets of size at most $\tau_p \cdot n$ and letting Γ_1 be the collection of all subsets of size at least $\tau_c \cdot n$. We refer to τ_p and τ_c as the privacy and correctness thresholds, respectively.

Ramp-threshold Secret Sharing. A ramp-threshold secret sharing scheme is a pair of PPT algorithms $(\mathsf{Share}, \mathsf{Reconst})$:

- $\mathsf{Share}(1^\lambda, n, \tau_p, \tau_c, s) \to (\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$ is the randomized secret sharing algorithm. It takes as input the security parameter 1^λ, the number of parties n, the privacy and correctness thresholds, and the secret $s \in S_\lambda$ where $S = \{S_\lambda\}$ is the space of secrets, and outputs the secret shares of all n parties.
- $\mathsf{Reconst}(\{\mathsf{sh}_i\}_{i \in \mathcal{S}}) \to s$ is the deterministic algorithm which takes as input the secret shares of a set $\mathcal{S} \subseteq [n]$ and outputs either the secret s or $\perp$.

A ramp-threshold secret sharing scheme must satisfy correctness as defined below:

Definition 16 (Correctness). *A ramp-threshold secret sharing scheme* $(\mathsf{Share}, \mathsf{Reconst})$ *satisfies correctness if for all* $\lambda \in \mathbb{N}$*, all* $0 < \tau_p < \tau_c \leq 1$*, all* $n \in \mathbb{N}$*, all secrets* $s \in S_\lambda$ *and every subset* $\mathcal{S} \subseteq [n]$ *of size* $\geq \tau_c n$*, it holds that,*

$$\Pr[\mathsf{Reconst}(\{\mathsf{sh}_i\}_{i\in\mathcal{S}}) = s] \geq 1 - \mathsf{negl}(\lambda),$$

where $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow\!\!\$\ \mathsf{Share}(1^\lambda, n, \tau_p, \tau_c, s)$*.*

As mentioned in Section 2.2.2, we additionally require the secret sharing scheme to satisfy share uniformity [8]. We formally define this property below:

Definition 17 (Share Uniformity). *A ramp-threshold secret sharing scheme* $(\mathsf{Share}, \mathsf{Reconst})$ *satisfies share uniformity if for all* $\lambda \in \mathbb{N}$*, all* $n \in \mathbb{N}$*, all* $0 < \tau_p < \tau_c \leq 1$*, for all secrets* $s \in S_\lambda$*, and any subset* $\mathcal{T} \subseteq [N]$ *such that* $|\mathcal{T}| \leq \tau_p n$*,*

$$(\{\mathsf{sh}_i\}_{i\in\mathcal{T}}) \equiv \left(\{\mathsf{sh}'_i\}_{i\in\mathcal{T}}\right),$$

where $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow\!\!\$\ \mathsf{Share}(1^\lambda, n, \tau_p, \tau_c, s)$*,* $\mathsf{sh}'_i \leftarrow\!\!\$\ (S_\lambda)^v$ *for all* $i \in \mathcal{T}$ *and* $v = v(\lambda)$ *denotes an upper bound on the size of the secret shares.*

Share uniformity also implies secrecy, hence we do not define it explicitly.

Theorem 6 (Theorem 1.1 and Remark 4.6 in [6]). *For every constants* $0 < \tau_p < \tau_c < 1$*, there exists a black-box* (τ_p, τ_c)*-ramp-threshold secret sharing scheme, with the following syntax:*

- *The* Share *algorithm is linear. The resulting secret share of each party* sh_i *consists of two parts:* $(\mathsf{psh}_i, \hat{\mathsf{sh}}_i := (\hat{\mathsf{sh}}_{i,1}, \ldots, \hat{\mathsf{sh}}_{i,y}))$*, where* psh_i *is a share of the public information, and* $\hat{\mathsf{sh}}_i$ *is the secret share.*
- *The* $\mathsf{Reconst}$ *algorithm is divided into two sub-routines:*
  - $\mathsf{Reconst}_1(\{\mathsf{psh}_i\}_{i\in\mathcal{S}}) \to \mathsf{psh}$ *takes as input the shares of public information and outputs* psh*.*
  - $\mathsf{Reconst}_2(\mathsf{psh}, \{\hat{\mathsf{sh}}_i\}_{i\in\mathcal{S}}) \to s$ *takes as input the reconstructed public information* psh *and shares* $\{\hat{\mathsf{sh}}_i\}$*, and outputs the secret. This algorithm makes only* $O(n)$ *additions with respect to the input shares* $\{\hat{\mathsf{sh}}_i\}$*. More formally, there exist reconstruction coefficients* $\{\nu_{i,j}\}_{i\in\mathcal{S},j\in[y]}$ *(which depend on* psh *and the set* $\mathcal{S}$*), such that the output of* $\mathsf{Reconst}$ *is equal to* $\Sigma_{i\in\mathcal{S},j\in[y]}\nu_{i,j} \cdot \hat{\mathsf{sh}}_{i,j}$*.*
  - *The reconstruction coefficients are small. Formally, for any authorized set* $\mathcal{S} \subseteq [n]$*, for all* $i \in \mathcal{S}$ *and all* $j \in [y]$*,* $|\nu_{i,j}| \leq \nu(n)$ *where* ν *is a polynomial.*
- *Moreover, the size of every secret share is* $O(1)$*, i.e. independent of the number of parties.*

Remark 2 (On Share Uniformity). The ramp-threshold scheme described in Theorem 6 relies on (k, n)-linear erasure codes with small coefficient decoding, which means that the decoding algorithm simply computes linear combinations of the codeword symbols with small coefficients. It is easy to see that if the erasure code also satisfies a linear independence property, which says that any subset of $\tau_p n$ rows of the generator matrix of the code are linearly independent, then the resulting secret sharing scheme achieves share uniformity. While we are currently not aware of a linear erasure code that satisfies both these properties, we conjecture that such codes do indeed exist. We leave the task of constructing such codes as an open problem.

6.3 Our Candidate Construction

Figures 8 and 9 present our candidate ABTE scheme based on lattices. We present it as a generic framework relying on a secret sharing scheme SS; it can be instantiated with any (ramp)-threshold secret sharing scheme that achieves linearity, small reconstruction coefficients and share uniformity. The public key and ciphertext sizes of the resulting scheme only scale with the logarithm of the number of parties, and the secret key size is proportional to the secret share size of the secret sharing scheme.

We use χ to denote a $\chi_{\max}$-bounded noise distribution for which the (ℓ, m, q, χ)-LWE problem is hard. Our scheme is parametrized by this bound $\chi_{\max}$, and B_{sm}, which denotes the bound on the smudging noise in each decryption share.

$\underline{\mathsf{Setup}}(1^\lambda, 1^n, 1^{1/\tau_p}, 1^{1/\tau_c}, 1^z)$:

1. Sample $(\boldsymbol{A}, \boldsymbol{T_A}) \leftarrow\!\!\$\ \mathsf{TrapGen}(1^\ell, 1^m, q)$
2. Sample $\boldsymbol{D}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_z \leftarrow\!\!\$\ \mathbb{Z}_q^{\ell \times m}$. Sample
$$((\mathsf{psh}_1, \{\boldsymbol{D}_{1,j}\}_{j\in[y]}), \ldots, (\mathsf{psh}_n, \{\boldsymbol{D}_{n,j}\}_{j\in[y]})) \leftarrow\!\!\$\ \mathsf{SS.Share}(1^\lambda, n, \tau_p, \tau_c, \boldsymbol{D}).$$
3. Output $\mathsf{mpk} := (\boldsymbol{A}, \boldsymbol{D}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_z)$ and $\mathsf{msk} := \boldsymbol{T_A}, \{(\mathsf{psh}_i, \{\boldsymbol{D}_{i,j}\}_{j\in[y]})\}_{i\in[n]}$.

$\underline{\mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, i, f)}$:

1. Compute $\boldsymbol{B}_f \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_z)$.
2. For all $j \in [y]$, compute $\boldsymbol{R}_{i,j} \leftarrow\!\!\$\ \mathsf{SampleRight}(\boldsymbol{A}, \boldsymbol{T_A}, \boldsymbol{B}_f, \boldsymbol{D}_{i,j}, \sigma)$.
3. Output $\mathsf{sk}_f^i \leftarrow (\mathsf{psh}_i, (\boldsymbol{R}_{i,1}, \ldots, \boldsymbol{R}_{i,y}))$.

Fig. 8. The Setup and KeyGen algorithms for our ABTE scheme from lattices $\mathsf{LWE\text{-}ABTE}$. For any attribute x and policy circuit f, we say that x satisfies the policy if $f(x) = 0$. The Eval algorithms are as described in Lemma 2.

Parameters. We require the following inequalities to hold for our scheme to be correct and secure:
- $\frac{(2m\sigma(\alpha_{\mathcal{F}}+1)\chi_{max})}{B_{sm}} = \mathsf{negl}(\lambda)$.
- $\chi_{max} + yn\nu(n) \cdot (B_{sm} + 2m\sigma(\alpha_{\mathcal{F}} + 1)\chi_{max}) < q/4$.
- $\sigma = \omega(\alpha_{\mathcal{F}}\sqrt{\log m})$.

The above requirements imply that the modulus q must be super-polynomial in the security parameter λ, but crucially, it remains polynomial in n.

Correctness. Theorem 7 below proves correctness of our construction.

Theorem 7. *The attribute-based threshold encryption scheme* $\mathsf{LWE\text{-}ABTE}$ *has negligible correctness error.*

Proof (of Theorem 7). We start with some more preliminaries before presenting the proof.

Lemma 3 ([13]). *For integers $\ell, m, k, q, \sigma > 0$, matrices $\boldsymbol{A} \in \mathbb{Z}_q^{\ell\times m}$ and $\boldsymbol{U} \in \mathbb{Z}_q^{\ell\times k}$, if $\boldsymbol{R} \in \mathbb{Z}^{m\times k}$ is sampled from $\mathcal{D}_\sigma(\Lambda_q^{\boldsymbol{U}}(\boldsymbol{A}))$ and $\boldsymbol{S}$ is sampled uniformly in $\{\pm 1\}^{m\times m}$ then*
$$\|\boldsymbol{R}^\top\|_2 \le \sigma\sqrt{mk} \quad , \quad \|\boldsymbol{R}\|_2 \le \sigma\sqrt{mk} \quad , \quad \|\boldsymbol{S}\|_2 \le 20\sqrt{m}$$
with overwhelming probability in m.

$\underline{\mathsf{Enc}(\mathsf{mpk}, x, \mu)}$: // $x \in \mathbb{Z}_q^z$

1. Sample $\boldsymbol{s} \leftarrow\!\!\$\ \mathbb{Z}_q^\ell$ and error vectors $\boldsymbol{e}_0, \boldsymbol{e}_1 \leftarrow\!\!\$\ \chi^m$.
2. Sample $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_z \leftarrow\!\!\$\ \{\pm 1\}^{m \times m}$. Compute:

$$\boldsymbol{H} := (\boldsymbol{A} | x_1 \boldsymbol{G} + \boldsymbol{B}_1 | \ldots | x_z \boldsymbol{G} + \boldsymbol{B}_z) \in \mathbb{Z}_q^{\ell \times (z+1)m}$$
$$\boldsymbol{e} := (\boldsymbol{I}_m | \boldsymbol{S}_1 | \ldots | \boldsymbol{S}_z)^\top \boldsymbol{e}_0 \in \mathbb{Z}_q^{(z+1)m}.$$

3. Output $\mathsf{ct}_x := (\boldsymbol{H}^\top \boldsymbol{s} + \boldsymbol{e}, \boldsymbol{D}^\top \boldsymbol{s} + \boldsymbol{e}_1 + \lceil \frac{q}{2} \rceil \mu) \in \mathbb{Z}_q^{(z+2)m}$.

$\underline{\mathsf{Dec}(\mathsf{sk}_f^i, \mathsf{ct}_x)}$:

1. If $f(x) \neq 0$, then output $\perp$.
2. Otherwise, let $\mathsf{ct}_x = (c_{in}, c_1, \ldots, c_z, c_{out}) \in \mathbb{Z}_q^{(z+2)m}$.
3. Compute $c_f := \mathsf{Eval}_{\mathsf{ct}}(f, \{x_i, \boldsymbol{B}_i, c_i\}_{i \in [z]}) \in \mathbb{Z}_q^m$. Let $c'_f := (c_{in} | c_f)$.
4. For all $j \in [y]$, sample $\boldsymbol{e}_{i,j} \leftarrow\!\!\$\ [-B_{sm}, B_{sm}]^m$ and compute $d_{i,j} := \boldsymbol{R}_{i,j}^\top c'_f + \boldsymbol{e}_{i,j}$.
5. Output $d_i := (\mathsf{psh}_i, (d_{i,1}, \ldots, d_{i,y}))$.

$\underline{\mathsf{Combine}(\mathsf{ct}_x, \{d_i\}_{i \in \mathcal{S}})}$:

1. Parse d_i as $\mathsf{psh}_i, (d_{i,1}, \ldots, d_{i,y})$.
2. Compute $\mathsf{psh} \leftarrow \mathsf{SS.Reconst}_1(\{\mathsf{psh}_i\}_{i \in \mathcal{S}})$.
3. Compute reconstruction coefficients $\{\nu_{i,j}\}_{i \in \mathcal{S}, j \in [y]}$ as in $\mathsf{SS.Reconst}_2$.
4. Output $\mathsf{Round}(c_{out} - \Sigma_{i \in \mathcal{S}, j \in [y]} \nu_{i,j} d_{i,j})$.

Fig. 9. The $\mathsf{Enc}, \mathsf{Dec}$ and $\mathsf{Combine}$ algorithms for our Attribute-based Threshold Encryption scheme from lattices LWE-ABTE.

For any attribute $x \in \mathbb{Z}_q^z$ and message $\mu \in \{0,1\}^m$, let ct_x denote the ciphertext. Let us consider a subset $\mathcal{S} \subseteq [n]$ of size $\tau_c n$, with $\{f_i\}_{i \in \mathcal{S}}$ as the corresponding policy circuits, such that $f_i(x) = 0$ for all $i \in \mathcal{S}$.

Then, for any $i \in \mathcal{S}$, by the properties of $\mathsf{Eval}_{\mathsf{ct}}$ listed above, we know that

$$c_f = (f(x)\boldsymbol{G} + \boldsymbol{B}_f)^\top \boldsymbol{s} + \hat{\boldsymbol{e}} = \boldsymbol{B}_f^\top \boldsymbol{s} + \hat{\boldsymbol{e}},$$

where $\|\hat{\boldsymbol{e}}\| < \Delta$. Consequently,

$$c'_f = (c_{in} | c_f) = (\boldsymbol{A} | \boldsymbol{B}_f)^\top \boldsymbol{s} + \boldsymbol{e}',$$

where $\|\boldsymbol{e}'\| < \Delta + \chi_{max} < (\alpha_{\mathcal{F}} + 1)\chi_{max}$. Next, observe that,

$$d_{i,j} = \boldsymbol{R}_{i,j}^\top \left((\boldsymbol{A} | \boldsymbol{B}_f)^\top \boldsymbol{s} + \boldsymbol{e}' \right) + \boldsymbol{e}_{i,j} = \boldsymbol{D}_{i,j}^\top \boldsymbol{s} + \boldsymbol{R}_{i,j}^\top \boldsymbol{e}' + \boldsymbol{e}_{i,j}.$$

By Lemmas 3 and 1, $\|\boldsymbol{R}_{i,j}\|_2 < 2m\sigma$. Hence,

$$\|\boldsymbol{R}_{i,j}^\top \boldsymbol{e}' + \boldsymbol{e}_{i,j}\| \leq 2m\sigma \cdot (\alpha_{\mathcal{F}} + 1)\chi_{max} + B_{sm}.$$

Let us now analyze the output of $\mathsf{Combine}$:

$$\begin{aligned} c_{out} - \Sigma_{i \in \mathcal{S}, j \in [y]} \nu_{i,j} d_{i,j} &= \boldsymbol{D}^\top \boldsymbol{s} + \boldsymbol{e}_1 + \lceil \frac{q}{2} \rceil \mu - \Sigma_{i \in \mathcal{S}, j \in [y]} \nu_{i,j} d_{i,j} \\ &= \boldsymbol{D}^\top \boldsymbol{s} + \boldsymbol{e}_1 + \lceil \frac{q}{2} \rceil \mu - \Sigma_{i \in \mathcal{S}, j \in [y]} \nu_{i,j} (\boldsymbol{D}_{i,j}^\top \boldsymbol{s} + \boldsymbol{R}_{i,j}^\top \boldsymbol{e}' + \boldsymbol{e}_{i,j}) \\ &= \lceil \frac{q}{2} \rceil \mu + \boldsymbol{e}_1 - \Sigma_{i \in \mathcal{S}, j \in [y]} \nu_{i,j} (\boldsymbol{R}_{i,j}^\top \boldsymbol{e}' + \boldsymbol{e}_{i,j}) \end{aligned}$$

To see why correctness holds, we observe that,

$$\|\boldsymbol{e}_1 - \Sigma_{i\in\mathcal{S},j\in[y]}\nu_{i,j}(\boldsymbol{R}_{i,j}^\top \boldsymbol{e}' + \boldsymbol{e}_{i,j})\| \leq \chi_{max} + \tau_c n y \nu(n)\cdot(2m\sigma\cdot(\alpha_{\mathcal{F}}+1)\chi_{max} + B_{sm}).$$

Since this noise term is bounded by $q/4$ (due to our parameter requirements listed above), the $\mathsf{Combine}$ algorithm does indeed output the correct decryption μ.

Security. Theorem 8 below proves CPA security of $\mathsf{LWE\text{-}ABTE}$ based on the LWE assumption, the well-formedness of preimage property along with linearity and share uniformity of the secret sharing scheme.

Theorem 8. *If the secret sharing scheme* SS *achieves linearity and share uniformity, then, for any* PPT *adversary* $\mathcal{A}$*, there exist* PPT *adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that*

$$\mathsf{Adv}^{\mathrm{sel-cpa}}_{\mathcal{A},\mathsf{LWE\text{-}ABTE}}(\lambda) \leq \mathsf{Adv}^{\mathrm{preimg,q},\sigma}_{\mathsf{TrapGen},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\ell,\mathrm{m,q},\chi}_{\mathcal{B}_2}(\lambda).$$

Proof (of Theorem 8). We start with some preliminaries, before presenting the proof:

Lemma 4. *Let* $\ell, m, q > 0$ *be integers with* q *prime. There exists a polynomial time algorithm* $\mathsf{SampleLeft}(\boldsymbol{A}, \boldsymbol{S}, y, \boldsymbol{U}, \sigma)$ *that, given full-rank matrix* $\boldsymbol{A} \in \mathbb{Z}_q^{\ell\times m}$*, matrices* $\boldsymbol{S}, \boldsymbol{U} \in \mathbb{Z}_q^{\ell\times m}$*,* $y \neq 0 \in \mathbb{Z}_q$ *and* $\sigma = \sqrt{5}\cdot(1+\|\boldsymbol{S}\|_2)\cdot\omega(\sqrt{\log m})$*, outputs a random sample* $\boldsymbol{X} \in \mathbb{Z}_q^{m\times m}$ *from a distribution that is statistically close to* $\mathcal{D}_\sigma(\Lambda_q^{\boldsymbol{U}}((\boldsymbol{A}|y\boldsymbol{G}+\boldsymbol{A}\boldsymbol{S})))$*, where* $\boldsymbol{G}$ *is the matrix defined above.*

Lemma 5 (Left-over Hash Lemma [2]). *Suppose that* $m > (\ell+1)\log_2 q + \omega(\log \ell)$ *and that* $q > 2$ *is prime. Let* $\boldsymbol{S}$ *be an* $m \times k$ *matrix chosen uniformly in* $\{\pm 1\}^{m\times k} \mod q$ *where* $k = k(\ell)$ *is polynomial in* ℓ*. Let* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *be matrices chosen uniformly in* $\mathbb{Z}_q^{\ell\times m}$ *and* $\mathbb{Z}_q^{\ell\times k}$ *respectively. Then, for all vectors* $\boldsymbol{e}$ *in* $\mathbb{Z}_q^n$*, the distribution* $(\boldsymbol{A}, \boldsymbol{A}\boldsymbol{S}, \boldsymbol{S}^\top\boldsymbol{e})$ *is statistically close to the distribution* $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{S}^\top\boldsymbol{e})$*.*

Note that the lemma holds for every vector $\boldsymbol{e}$ in $\mathbb{Z}_q^m$, including low norm vectors.

Definition 18 (Statistical Distance). *Let* E *be a finite set,* Ω *a probability space, and* $X, Y : \Omega \to E$ *random variables. We define the* statistical distance *between* X *and* Y *to be the function* Δ *defined by*

$$\Delta(X,Y) = \frac{1}{2}\sum_{e\in E}\left|\Pr[X=e] - \Pr[Y=e]\right|.$$

Lemma 6 (Smudging Lemma). *Let* $B_1, B_2 \in \mathbb{N}$*. For any* $e_1 \in [-B_1, B_1]$*, let* E_1 *and* E_2 *be independent random variables uniformly distributed in* $[-B_2, B_2]$ *and define two stochastic variables* $X_1 = E_1 + e_1$ *and* $X_2 = E_2$*. Then* $\Delta(X_1, X_2) < B_1/B_2$*.*

We prove the theorem using a series of games where the first game is identical to the security game given in Definition 10. In the last game in the sequence the adversary has advantage zero. We show that a PPT adversary cannot distinguish between the games which will prove that the adversary has negligible advantage in winning the original ABE security game. We use the well-sampledness property in proving that Games 1 and 2 are indistinguishable, and the LWE problem in proving that Games 2 and 3 are indistinguishable.

Game 0. This is the original security game. Recall that the game starts with the adversary sending the challenge attribute x^* to the challenger. The challenger then samples the public parameters

including matrices $\boldsymbol{A}, \boldsymbol{D}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_z$. The adversary then queries upto $\tau_p n$ (where τ_p is the privacy threshold) secret keys for parties with policies satisfied by x^*. It can then query keys for any party for non-satisfying policies, and also ask for decryption shares for any party for any valid ciphertext. We use ct^* to denote the challenge ciphertext, and use $\boldsymbol{S}_1^*, \ldots, \boldsymbol{S}_z^*$ to denote the random matrices used to produce ct^*.

Game 1. We change how the challenger samples the matrices $\{\boldsymbol{B}_i\}$. In particular, the challenger first samples matrices $\boldsymbol{S}_1^*, \ldots, \boldsymbol{S}_z^* \leftarrow\!\!\$\ \pm 1^{m \times m}$, and then sets

$$\boldsymbol{B}_i = \boldsymbol{A}\boldsymbol{S}_i^* - x_i^* \boldsymbol{G} \ \forall\ i \in [z].$$

The rest of the game remains unchanged.

We show that Game 0 is statistically indistinguishable from Game 1 by Lemma 5. Observe that in Game 1 the matrices $\boldsymbol{S}_i^*$ are used only in the construction of $\boldsymbol{B}_i$ and in the construction of the challenge ciphertext where $\boldsymbol{e} = (\boldsymbol{I}_m | \boldsymbol{S}_1^* | \ldots | \boldsymbol{S}_z^*)^\top \cdot \boldsymbol{e}_0$ is used as the noise vector for some $\boldsymbol{e}_0 \in \mathbb{Z}_q^m$. Let $\boldsymbol{S}^* = (\boldsymbol{S}_1^* | \ldots | \boldsymbol{S}_z^*)$, then by Lemma 5 the distribution $(\boldsymbol{A}, \boldsymbol{A}\boldsymbol{S}^*, \boldsymbol{e})$ is statistically close to the distribution $(\boldsymbol{A}, \boldsymbol{A}', \boldsymbol{e})$ where $\boldsymbol{A}'$ is a uniform matrix in $\mathbb{Z}_q^{\ell \times zm}$. It follows that in the adversary's view, all the matrices $\boldsymbol{A}\boldsymbol{S}_i^*$ are statistically close to uniform and therefore the $\boldsymbol{B}_i$ as defined above are close to uniform. Hence, the $\boldsymbol{B}_i$ in Games 0 and 1 are statistically indistinguishable.

Game 2. We now change how the matrix $\boldsymbol{A}$ is chosen in mpk. In game 2, we generate $\boldsymbol{A}$ as a uniform random matrix in $\mathbb{Z}_q^{\ell \times m}$. The construction of $\boldsymbol{B}_1, \ldots, \boldsymbol{B}_z$ remains the same as in Game 1 above. We maintain a set $\mathcal{J} := \emptyset$ storing the list of all parties whose keys have been queried by $\mathcal{A}$ for satisfying policies. We also maintain a map $M : [n] \to \mathcal{F}$ to store the satisfying policies that the adversary queries the keys for. We now describe how we respond to all the oracle queries by $\mathcal{A}$:

- $\mathsf{sk}(i, f_i)$ **such that** $f_i(x^*) = 0$**.** If $i \in \mathcal{J}$ or if $i \notin \mathcal{J}$ but $|\mathcal{J}| = \tau_p n$, we simply abort (note that this is identical to the challenger in Game 1). Otherwise, we sample $\boldsymbol{R}_{i,j} \leftarrow\!\!\$\ \mathcal{D}_{\mathbb{Z},\sigma}^{2m \times m}$ for all $j \in [y]$. We then set $\boldsymbol{D}_{i,j} = (\boldsymbol{A} | \boldsymbol{B}_{f_i}) \boldsymbol{R}_{i,j}$, where $\boldsymbol{B}_{f_i} = \mathsf{Eval}_{\mathsf{pk}}(f_i, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_z)$, set $\mathcal{J} \leftarrow \mathcal{J} \cup \{i\}$ and $M[i] = f_i$. We also sample psh_i uniformly randomly from the share space. We then return $\mathsf{sk}_{f_i}^i = \mathsf{psh}_i, (\boldsymbol{R}_{i,1}, \ldots, \boldsymbol{R}_{i,y})$ to $\mathcal{A}$. Below, we will use well-sampledness to argue that these keys are computationally indistinguishable from the real keys.
  Note that since we are considering the selective game, the adversary must make all the key queries for satisfying policies before making key queries for non-satisfying policies or decryption queries. After the adversary is done with making all such queries i.e. keys for satisfying policies, we first sample the remaining shares of $\boldsymbol{D}$. More formally, we run the Share algorithm with inputs $1^\lambda, n, \tau_p, \tau_c, \boldsymbol{D}$, conditioned on $\mathsf{sh}_i = \mathsf{psh}_i, (\boldsymbol{D}_{i,1}, \ldots, \boldsymbol{D}_{i,y})$ for all $i \in \mathcal{J}$. This is possible since $|\mathcal{J}| \leq \tau_p n$, and the sharing algorithm is linear.
- $\mathsf{sk}(i, f_i)$ **such that** $f_i(x^*) \neq 0$**.** We will use the trapdoor $\boldsymbol{T}_{\boldsymbol{G}}$ to generate the key for (i, f_i), as follows. Let $\boldsymbol{B}_{f_i} = \mathsf{Eval}_{\mathsf{pk}}(f_i, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_z)$. We need to find matrices $\boldsymbol{R}_{i,j}$ such that

  $$(\boldsymbol{A} | \boldsymbol{B}_{f_i}) \boldsymbol{R}_{i,j} = \boldsymbol{D}_{i,j}.$$

  To do so, we run $\boldsymbol{S}_{f_i} \leftarrow \mathsf{Eval}_{\mathsf{sim}}(f_i, ((x_i^*, \boldsymbol{S}_i^*)_{i=1}^z, \boldsymbol{A})$ and obtain a low-norm matrix $\boldsymbol{S}_{f_i} \in \mathbb{Z}_q^{m \times m}$ such that $\boldsymbol{A}\boldsymbol{S}_{f_i} - f_i(x^*)\boldsymbol{G} = \boldsymbol{B}_{f_i}$. By definition of $\mathsf{Eval}_{\mathsf{sim}}$ we know that $\|\boldsymbol{S}_{f_i}\|_2 \leq \alpha_{\mathcal{F}}$.
  Finally, we respond with $\boldsymbol{R}_{i,j} = \mathsf{SampleLeft}(\boldsymbol{A}, \boldsymbol{S}_{f_i}, -f_i(x^*), \boldsymbol{D}_{i,j}, \sigma)$ for all $j \in [y]$. By definition of $\mathsf{SampleLeft}$, we know that $\boldsymbol{R}_{i,j}$ is distributed as required. Indeed because $\|\boldsymbol{S}_{f_i}\|_2 \leq \alpha_{\mathcal{F}}(\ell)$, $\sigma = \sqrt{5} \cdot (1 + \|\boldsymbol{S}_f\|_2) \cdot \omega(\sqrt{\log m})$ as needed for algorithm $\mathsf{SampleLeft}$ in Lemma 4. We return $\mathsf{psh}_i, (\boldsymbol{R}_{i,1}, \ldots, \boldsymbol{R}_{i,y})$ to $\mathcal{A}$.

– $\mathsf{dec}(\mu, r, x, i, f_i)$**.** If $f_i(x) \neq 0$, we simply output $\perp$. Otherwise,
  - If $i \in \mathcal{J}$ and $f_i = M[i]$: then we already know the secret key $\mathsf{sk}^i_{f_i}$. We simply output $\mathsf{Dec}(\mathsf{sk}^i_{f_i}, \mathsf{ct}_x)$ where $\mathsf{ct}_x = \mathsf{Enc}(\mathsf{mpk}, x, \mu; r)$.
  - Otherwise parse the ciphertext randomness r as $\boldsymbol{s}, \boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{S}_1, \ldots, \boldsymbol{S}_z$. For each $j \in [y]$, we compute $d'_{i,j} = \boldsymbol{D}^\top_{i,j}\boldsymbol{s} + \boldsymbol{e}'_{i,j}$ where $\boldsymbol{e}'_{i,j} \leftarrow\!\$\, [-B_{sm}, B_{sm}]^m$. We simply output $\mathsf{psh}_i, (d'_{i,1}, \ldots, d'_{i,y})$. We argue that these shares are distributed identically distributed as the real shares. To see that, recall that by correctness, the real decryption shares $d_{i,j}$ are equal to:

$$d_{i,j} = \boldsymbol{D}^\top_{i,j}\boldsymbol{s} + \boldsymbol{R}^\top_{i,j}\boldsymbol{e}' + \boldsymbol{e}_{i,j}$$

    where $\|\boldsymbol{R}^\top_{i,j}\boldsymbol{e}'\| \leq 2m\sigma \cdot (\alpha_{\mathcal{F}} + 1)\chi_{max}$ and $\boldsymbol{e}_{i,j}$ is also sampled uniformly randomly from $[-B_{sm}, B_{sm}]^m$. Since our parameters are such that this value is equal to $\mathsf{negl}(\lambda) \cdot B_{sm}$, Lemma 6 implies that $d_{i,j}$ and $d'_{i,j}$ are identically distributed.

So far, we have argued that the responses to all oracle queries (except for keys for satisfying policies) are statistically close to the responses given in Game 1. We now argue that the responses to key queries for satisfying policies are computationally close to those in Game 1, using well-sampledness. Formally, we will make an adversary $\mathcal{B}_1$ which can break well-sampledness if $\mathcal{A}$ can distinguish between the two games.

Reduction from well-sampledness. $\mathcal{B}_1$ acts as the challenger to $\mathcal{A}$, and simulates either Game 1 or 2. $\mathcal{B}_1$ gets as input x^* from $\mathcal{A}$. It forwards the parameters ℓ, m to its challenger, and gets back a matrix $\boldsymbol{A}$. $\mathcal{B}_1$ then generates the public parameters as in Game 2. Specifically, it samples matrices $\boldsymbol{S}^*_1, \ldots, \boldsymbol{S}^*_z \leftarrow\!\$\, \{\pm 1\}^{m \times m}$, sets $\boldsymbol{B}_i = \boldsymbol{A}\boldsymbol{S}^*_i - x^*_i\boldsymbol{G}$, and samples $\boldsymbol{D}$ randomly. $\mathcal{B}_1$ sends these parameters to $\mathcal{A}$, and responds to oracle queries as follows:

– $\mathsf{sk}(i, f_i)$ **such that** $f_i(x^*) = 0$**.** If $i \in \mathcal{J}$ or if $i \in \mathcal{J}$ but $|\mathcal{J}| = \tau_p n$, then $\mathcal{B}_2$ simply aborts. Otherwise, for each $j \in [y]$, $\mathcal{B}_1$ sends m queries to its well-sampledness challenger, say the responses are $\boldsymbol{x}_{i,j,1}, \ldots, \boldsymbol{x}_{i,j,m}$. It then samples vectors $\boldsymbol{v}_{i,j,1}, \boldsymbol{v}_{i,j,m}$ from the distribution $\mathcal{D}^m_{\mathbb{Z},\sigma}$, and sets $\boldsymbol{R}_{i,j}$ to be:

$$\boldsymbol{R}_{i,j} = \begin{bmatrix} \boldsymbol{x}_{i,j,1} \ldots \boldsymbol{x}_{i,j,m} \\ \boldsymbol{v}_{i,j,1} \ldots \boldsymbol{v}_{i,j,m} \end{bmatrix}$$

– $\mathcal{B}_1$ responds to all other queries exactly as in Game 2.
$\mathcal{B}_1$ samples the challenge ciphertext as in Games 1 and 2. We argue that, when the bit b sampled by the well-sampledness challenger is 0, i.e. $\boldsymbol{x}_{i,j,w}$ is the pre-image of a uniformly random vector $\boldsymbol{u}_{i,j,w}$ with respect to $\boldsymbol{A}$, hence, $(\boldsymbol{A}|\boldsymbol{B}_{f_i})\boldsymbol{R}_{i,j} = [\boldsymbol{u}_{i,j,1} + \boldsymbol{B}_{f_i}\boldsymbol{v}_{i,j,1} \ldots \boldsymbol{u}_{i,j,n} + \boldsymbol{B}_{f_i}\boldsymbol{v}_{i,j,n}]$ is also uniformly random. This is identically distributed to Game 1, where the shares $\{\boldsymbol{D}_{i,j}\}_{i \in \mathcal{S}, j \in [y]}$ values for any set $\mathcal{S} \subseteq [n]$ of size $\tau_p n$ is distributed uniformly, by share uniformity of the secret sharing scheme. Moreover, as already argued above, the public parameters and the responses to all other oracles are already distributed identically in Games 1 and 2.
On the other hand, if the bit b sampled by the well-sampledness challenger is 1, then the whole matrix $\boldsymbol{R}_{i,j}$ as constructed above, is sampled from the gaussian distribution $\mathcal{D}^{2m \times m}_{\mathbb{Z},\sigma}$. Hence, in this case, $\mathcal{B}_1$ has perfectly simulated Game 2.
Finally, $\mathcal{A}$ guesses if it is interacting with a Game 1 or Game 2 challenger. $\mathcal{B}_1$ simply outputs $\mathcal{A}$'s guess as the answer. Since $\mathcal{B}_1$ perfectly simulates Games 1 and 2, $\mathcal{B}_1$'s advantage in solving well-sampledness is the same as $\mathcal{A}$'s advantage in distinguishing Games 1 and 2, as required. This completes the description of $\mathcal{B}_1$.

Game 3. This game is identical to Game 2 except that the challenge ciphertext ct^* is now sampled uniformly at random from $\mathbb{Z}_q^{(z+2)m}$. Since the challenge ciphertext is always a fresh random element in the ciphertext space, $\mathcal{A}$'s advantage in this game is zero. It remains to show that Game 2 and Game 3 are computationally indistinguishable for a PPT adversary, which we do by giving a reduction from the LWE problem.

Reduction from LWE. We will make an LWE adversary $\mathcal{B}_2$ that acts as the challenger to $\mathcal{A}$. $\mathcal{B}_2$ begins by obtaining an LWE challenge consisting of two random matrices $\boldsymbol{A}, \boldsymbol{D}$ in $\mathbb{Z}_q^{\ell \times m}$ and two vectors $c_{in}, c_{out} \in \mathbb{Z}_q^m$. We know that c_{in}, c_{out} are either random in $\mathbb{Z}_q^m$ or

$$c_{in} = \boldsymbol{A}^\top \boldsymbol{s} + \boldsymbol{e}_0 \quad \text{and} \quad c_{out} = \boldsymbol{D}^\top \boldsymbol{s} + \boldsymbol{e}_1$$

for some random vector $\boldsymbol{s} \in \mathbb{Z}_q^\ell$ and $\boldsymbol{e}_0, \boldsymbol{e}_1 \leftarrow\!\!\$\ \chi^m$. We show how $\mathcal{B}_2$ can distinguish between these cases using $\mathcal{A}$:

The game begins with $\mathcal{A}$ sending the challenge attribute x^* to $\mathcal{B}_2$. $\mathcal{B}_2$ generates public parameters as in Game 2 above: choose random matrices $\boldsymbol{S}_i^* \leftarrow\!\!\$\ \{\pm 1\}^{m \times m}$ and set $\boldsymbol{B}_i = \boldsymbol{A}\boldsymbol{S}_i^* - x_i^* \boldsymbol{G}$ for all $i \in [z]$. It sets $\mathsf{mpk} = (\boldsymbol{A}, \boldsymbol{D}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_z)$ and sends this to $\mathcal{A}$. $\mathcal{B}_2$ responds to all oracle queries exactly as in Game 2, as described above. Eventually, when $\mathcal{B}_2$ receives μ_0, μ_1 from $\mathcal{A}$, it prepares the challenge ciphertext by sampling $b \leftarrow\!\!\$\ \{0,1\}$, and computing,

$$c_0^* = (\boldsymbol{I}_m | \boldsymbol{S}_1^* | \ldots | \boldsymbol{S}_z^*)^\top c_{in} \in \mathbb{Z}_q^{(z+1)m},$$

and $c^* = (c_0^*, c_{out} + \lceil \frac{q}{2} \rceil \mu_b)$. $\mathcal{B}_2$ sends c^* as the ciphertext to $\mathcal{A}$. We argue that, when the LWE challenge is pseudorandom, then c^* is distributed exactly as in Game 2. To see why, observe that when encrypting with respect to x^*, the matrix $\boldsymbol{H}$ is as follows:

$$\begin{aligned} \boldsymbol{H} &= \boldsymbol{A} | x_1^* \boldsymbol{G} + \boldsymbol{B}_1 | \ldots | x_z^* \boldsymbol{G} + \boldsymbol{B}_z \\ &= \boldsymbol{A} | x_1^* \boldsymbol{G} + \boldsymbol{A}\S_1^* - x_1^* \boldsymbol{G} | \ldots | x_z^* \boldsymbol{G} + \boldsymbol{A}\boldsymbol{S}_z^* - x_z^* \boldsymbol{G} \\ &= \boldsymbol{A} | \boldsymbol{A}\S_1^* | \ldots | \boldsymbol{A}\boldsymbol{S}_z^* \end{aligned}$$

Therefore, c_0^* as defined above is equal to:

$$\begin{aligned} c_0^* &= (\boldsymbol{I}_m | \boldsymbol{S}_1^* | \ldots | \boldsymbol{S}_z^*)^\top (\boldsymbol{A}^\top \boldsymbol{s} + \boldsymbol{e}_0) \\ &= (\boldsymbol{A} | \boldsymbol{A}\S_1^* | \ldots | \boldsymbol{A}\boldsymbol{S}_z^*)^\top \boldsymbol{s} + (\boldsymbol{I}_m | \boldsymbol{S}_1^* | \ldots | \boldsymbol{S}_z^*)^\top \boldsymbol{e}_0 \\ &= \boldsymbol{H}^\top \boldsymbol{s} + \boldsymbol{e} \end{aligned}$$

where $\boldsymbol{e} = (\boldsymbol{I}_m | \boldsymbol{S}_1^* | \ldots | \boldsymbol{S}_z^*)^\top \boldsymbol{e}_0$. This $\boldsymbol{e}$ is sampled from the same distribution as the noise vector $\boldsymbol{e}$ in algorithm Enc. We therefore conclude that c_0^* is computed as in Game 2. Moreover, since $c_{out} = \boldsymbol{D}^\top \boldsymbol{s} + \boldsymbol{e}_1$, we know that the entire challenge ciphertext c^* is a valid encryption of μ_b with respect to x^* as required.

When the LWE challenge is random, we know that c_{in} and c_{out} are uniform in $\mathbb{Z}_q^m$. Therefore the public parameters and c_0^* as constructed by $\mathcal{B}_2$ are uniform and independent in $\mathbb{Z}_q^{(z+1)m}$ by a standard application of the left over hash lemma (e.g. [13]) where the universal hash function is defined as multiplication by the random matrix $(\boldsymbol{A}^\top | c_{in})^\top$. Since c_{out} is also uniform, the challenge ciphertext overall is uniform in $\mathbb{Z}_q^{(z+2)m}$, as in Game 3.

Finally, $\mathcal{A}$ guesses if it is interacting with a Game 2 or Game 3 challenger. $\mathcal{B}_2$ outputs $\mathcal{A}$'s guess as the answer to the LWE challenge it is trying to solve. We already argued that when the

LWE challenge is pseudorandom the adversary's view is as in Game 2. When the LWE challenge is random, the adversary's view is as in Game 3. Hence, $\mathcal{B}_2$'s advantage in solving LWE is the same as $\mathcal{A}$'s advantage in distinguishing Games 2 and 3, as required. This completes the description of algorithm $\mathcal{B}_2$.

7 Threshold Traitor Tracing from Threshold PLBE

In this section, we present a generic compiler to construct Threshold Traitor Tracing from Threshold PLBE. We start with defining threshold PLBE in Section 7.1 before describing our compiler in Section 7.2.

7.1 Threshold Private Linear Broadcast Encryption

A Threshold Private Linear Broadcast Encryption is a tuple of six algorithms $\mathsf{TPLBE} = (\mathsf{Setup}, \mathsf{EncPK}, \mathsf{EncSK}, \mathsf{Dec}, \mathsf{Combine})$:

- $\mathsf{Setup}(1^\lambda, 1^n, 1^t) \to (\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]})$ is the probabilistic **setup** algorithm that takes as inputs the security parameter λ, the number of parties n and the threshold t, and outputs the public parameters pp, the master key pair $(\mathsf{mpk}, \mathsf{msk})$ and the n secret keys $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n$.
- $\mathsf{EncPK}(\mathsf{mpk}, \mu) \to \mathsf{ct}$ is the probabilistic **public key encryption** algorithm that takes as input the master public key mpk, a message μ and outputs a ciphertext ct.
- $\mathsf{EncSK}(\mathsf{msk}, \mu, j) \to \mathsf{ct}$ is the **secret key encryption** algorithm that takes as inputs the master secret key msk, a message μ, an index $j \in [n]$ and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) \to d_i$ is the deterministic **decryption** algorithm that takes as inputs a secret key sk_i, a ciphertext ct and outputs the partial decryption d_i of ct under sk_i.
- $\mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}, \{d_j\}_{j\in\mathcal{J}}) \to \mu$ is the deterministic **combiner** algorithm that takes as inputs, a combiner key pkc, a ciphertext ct, a set $\mathcal{J} \subseteq [n]$, a set of decryption shares $\{d_j\}_{j\in\mathcal{J}}$ and outputs either a message μ.

Definition 19 (Correctness). *A* TPLBE *scheme is said to be correct if there exists negligible functions* $\mathsf{negl}_1(\lambda)$, $\mathsf{negl}_2(\lambda)$ *such that, for all* $\lambda \in \mathbb{N}$, *all* $t < n \in \mathbb{N}$, *and for any subset* $\mathcal{S} \subseteq [n]$ *of size* t *and for any message* μ, *we have:*

$$\Pr\left[\mu' = \mu \;:\; \begin{array}{c}(\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow\$ \mathsf{Setup}(1^\lambda, 1^n, 1^t) \\ \mathsf{ct} \leftarrow\$ \mathsf{EncPK}(\mathsf{mpk}, \mu) \\ d_i := \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) \;\forall\; i \in \mathcal{S} \\ \mu' := \mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}, \mathcal{S}, \{d_j\}_{j\in\mathcal{S}})\end{array}\right] \geq 1 - \mathsf{negl}_1(\lambda)$$

Moreover, for all $j \in [n]$ *such that* $j < i$ *for all* $i \in \mathcal{S}$, *we have,*

$$\Pr\left[\mu' = \mu \;:\; \begin{array}{c}(\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow\$ \mathsf{Setup}(1^\lambda, 1^n, 1^t) \\ \mathsf{ct} \leftarrow\$ \mathsf{EncSK}(\mathsf{msk}, \mu, j) \\ d_i := \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) \;\forall\; i \in \mathcal{S} \\ \mu' := \mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}_x, \mathcal{S}, \{d_i\}_{i\in\mathcal{S}})\end{array}\right] \geq 1 - \mathsf{negl}_2(\lambda).$$

Semantic Security. We define semantic security for threshold PLBE which requires that any PPT adversary that corrupts up to $t-1$ parties, and can query partial decryptions of arbitrary ciphertexts, cannot learn anything about the message, from a ciphertext generated using $\mathsf{EncPK}(\mathsf{mpk}, \cdot)$.

We require this to hold even if the adversary is given access to msk, which allows it to compute encryptions for arbitrary indices $j \in \{0\} \cup [n]$. We formalize this using a game-based definition in Definition 20 and Figure 10. Similar to Section 3.1, we will focus on the semi-adaptive setting, where the adversary must query all secret keys before querying any partial decryption.

Definition 20 (CPA Security). *A* TPLBE *scheme is said to satisfy semi-adaptive strong CPA security if for every* PPT *adversary* $\mathcal{A}$*, the following function is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{sa-cpa}}_{\mathcal{A},\mathsf{TPLBE}}(\lambda) := \left|\frac{1}{2} - \Pr[\mathbf{IND\text{-}CPA}^{sa}_{\mathcal{A},\mathsf{TPLBE}}(\lambda) = 1]\right|.$$

Experiment $\mathbf{IND\text{-}CPA}_{\mathcal{A},\mathsf{TPLBE}}(\lambda)$

$\mathcal{J} \leftarrow \emptyset$
$(n, t, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$
$(\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, 1^n, 1^t)$
$(\mu_0, \mu_1, \mathsf{state}) \leftarrow\!\$\ \mathcal{A}^{\mathsf{sk}(\cdot),\mathsf{dec}(\cdot,\cdot,\cdot)}(\mathsf{state}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk})$
$b \leftarrow\!\$\ \{0,1\},\ \mathsf{ct} \leftarrow\!\$\ \mathsf{EncPK}(\mathsf{mpk}, \mu_b)$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathsf{sk}(\cdot),\mathsf{dec}(\cdot,\cdot,\cdot)}(\mathsf{state}, \mathsf{ct})$
if $|\mathcal{J}| \geq t$ **then return** 0
if $b' = b$ **then return** 1 **else return** 0

Oracle $\mathsf{sk}(i)$

if $i \notin [n]$ **then return** $\perp$
$\mathcal{J} \leftarrow \mathcal{J} \cup \{i\}$
return sk_i

Oracle $\mathsf{dec}(\mu, r, i)$

if $i \notin [n]$ **then return** $\perp$
return $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{EncPK}(\mathsf{mpk}, \mu; r))$

Fig. 10. The semantic security experiment for a threshold PLBE scheme TPLBE and an adversary $\mathcal{A}$.

q-Bounded Security. Similar to PLBE, the security of a threshold PLBE is captured through the properties of *indistinguishability*, *index hiding* and *message hiding*. The indistinguishability property ensures that any PPT adversary controlling an arbitrary number of parties cannot distinguish if a ciphertext ct was generated using the public key encryption mode EncPK or using the secret key encryption mode EncSK with index $j = 0$. Message-hiding property says that message privacy is preserved against an adversary controlling any number of parties for a ciphertext that has been encrypted using the secret key mode with index $j = n$. That is, an adversary holding any number of keys for parties that are not allowed to decrypt, cannot learn anything about the message. The index-hiding property ensures that ciphertexts encrypted using EncSK with index $j = i$ and those encrypted with index $j = i + 1$ are indistinguishable to an adversary that does not have the secret key sk_i. We formally define these properties below:

Definition 21 (q-bounded Indistinguishability). *A threshold PLBE scheme* TPLBE *is said to satisfy q-bounded indistinguishability if for any* PPT *adversary* $\mathcal{A}$ *that makes at most* q *many*

encryption queries to $\mathsf{EncSK}(\mathsf{msk},\cdot,0)$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[b'=b \;:\; \begin{array}{c}(n,t,\mathsf{state})\leftarrow\$\,\mathcal{A}(1^\lambda)\\ (\mathsf{pp},\mathsf{mpk},\mathsf{pkc},\mathsf{msk},\{\mathsf{sk}_i\}_{i\in[n]})\leftarrow\$\,\mathsf{Setup}(1^\lambda,1^n,1^t)\\ (\mu,\mathsf{state})\leftarrow\mathcal{A}^{\mathsf{EncSK}(\mathsf{msk},\cdot,0)}(\mathsf{state},\mathsf{mpk},\{\mathsf{sk}_i\}_{i\in[n]})\\ b\leftarrow\$\,\{0,1\}\\ \mathsf{ct}_0\leftarrow\$\,\mathsf{EncSK}(\mathsf{msk},\mu,0),\mathsf{ct}_1\leftarrow\$\,\mathsf{EncPK}(\mathsf{mpk},\mu)\\ b'\leftarrow\mathcal{A}^{\mathsf{EncSK}(\mathsf{msk},\cdot,0)}(\mathsf{state},\mathsf{ct}_b)\end{array}\right]\leq\frac{1}{2}+\mathsf{negl}(\lambda).$$

Note that here $\mathcal{A}$ *is only allowed to query for ciphertexts corresponding to index* 0*.*

Definition 22 (q-bounded Index Hiding Security). *A threshold PLBE scheme* TPLBE *is said to satisfy q-bounded index hiding security if for any* PPT *adversary* $\mathcal{A}$ *that makes* q *many encryption queries to* $\mathsf{EncSK}(\mathsf{msk},\cdot,\cdot)$ *and for all indices* $i^*\in\{0,\ldots,n-1\}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that the following probability is bounded by* $1/2+\mathsf{negl}(\lambda)$*:*

$$\Pr\left[b'=b \;:\; \begin{array}{c}(n,t,\mathsf{state})\leftarrow\$\,\mathcal{A}(1^\lambda)\\ (\mathsf{pp},\mathsf{mpk},\mathsf{pkc},\mathsf{msk},\{\mathsf{sk}_i\}_{i\in[n]})\leftarrow\$\,\mathsf{Setup}(1^\lambda,1^n,1^t)\\ (\mu,\mathsf{state})\leftarrow\mathcal{A}^{\mathsf{EncSK}(\mathsf{msk},\cdot,\cdot)}(\mathsf{state},\mathsf{mpk},\mathsf{pkc},\{\mathsf{sk}_i\}_{i\in[n]\setminus i^*})\\ b\leftarrow\$\,\{0,1\},\ \mathsf{ct}\leftarrow\$\,\mathsf{EncSK}(\mathsf{msk},\mu,i^*+b)\\ b'\leftarrow\mathcal{A}^{\mathsf{EncSK}(\mathsf{msk},\cdot)}(\mathsf{state},\mathsf{ct})\end{array}\right]$$

Definition 23 (q-bounded Message Hiding Security). *A threshold PLBE scheme* TPLBE *is said to satisfy q-bounded message hiding security if for any* PPT *adversary* $\mathcal{A}$ *that can submit* q *many encryption queries to* $\mathsf{EncSK}(\mathsf{msk},\cdot,\cdot)$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that the following probability is bounded by* $1/2+\mathsf{negl}(\lambda)$*:*

$$\Pr\left[b'=b \;:\; \begin{array}{c}(n,t,\mathsf{state})\leftarrow\$\,\mathcal{A}(1^\lambda)\\ (\mathsf{pp},\mathsf{mpk},\mathsf{pkc},\mathsf{msk},\{\mathsf{sk}_i\}_{i\in[n]})\leftarrow\$\,\mathsf{Setup}(1^\lambda,1^n,1^t)\\ (\mathsf{state},\mu_0,\mu_1)\leftarrow\mathcal{A}^{\mathsf{EncSK}(\mathsf{msk},\cdot,\cdot)}(\mathsf{state},\mathsf{mpk},\mathsf{pkc},\{\mathsf{sk}_i\}_{i\in[n]})\\ b\leftarrow\$\,\{0,1\},\ \mathsf{ct}\leftarrow\$\,\mathsf{EncSK}(\mathsf{msk},\mu_b,n)\\ b'\leftarrow\mathcal{A}^{\mathsf{EncSK}(\mathsf{msk},\cdot)}(\mathsf{state},\mathsf{ct})\end{array}\right]$$

Decoder-based Security. As observed in [46], we also need to consider decoder-based security definitions for traceability purposes. We start with defining the notion of *distinguishing decoders* for TPLBE with respect to different encryption modes.

Definition 24 (TPLBE Distinguisher). *For any* $\gamma\in\left[-\frac{1}{2},\frac{1}{2}\right]$*, a* PPT *decoder* D*, for* $\mathsf{params}\leftarrow\mathsf{Setup}(1^\lambda,1^n,1^t)$ *and for any message* μ *is said to be*

γ-$\mathsf{Dist}^{\mathsf{pk},(\mathsf{sk},0)}_{\mathsf{params}}$ *if*

$$\Pr\left[b'=b \;:\; \begin{array}{c}b\leftarrow\$\,\{0,1\}\\ \mathsf{ct}_0\leftarrow\$\,\mathsf{EncSK}(\mathsf{msk},\mu,0),\mathsf{ct}_1\leftarrow\$\,\mathsf{EncPK}(\mathsf{mpk},\mu)\\ b'\leftarrow D(\mathsf{ct}_b)\end{array}\right]\geq\frac{1}{2}+\gamma$$

γ-$\mathsf{Dist}^{i,i+1}_{\mathsf{params}}$ *if for any index* i*,*

$$\Pr\left[b'=b \;:\; \begin{array}{c}b\leftarrow\$\,\{0,1\},\ \mathsf{ct}\leftarrow\$\,\mathsf{EncSK}(\mathsf{msk},\mu,i+b)\\ b'\leftarrow D(\mathsf{ct})\end{array}\right]\geq\frac{1}{2}+\gamma$$

$\gamma\text{-}\mathsf{Dist}^{n}_{\mathsf{params}}$ *for any two messages* μ_0, μ_1 *if,*

$$\Pr\left[b' = b \ : \ \begin{array}{c} b \leftarrow\$ \{0,1\},\ \mathsf{ct} \leftarrow\$ \mathsf{EncSK}(\mathsf{msk}, \mu_b, n) \\ b' \leftarrow D(\mathsf{ct}) \end{array}\right] \geq \frac{1}{2} + \gamma$$

Definition 25 (Decoder-based Indistinguishability). *A* TPLBE *scheme is said to satisfy decoder-based indistinguishability if for any* PPT *adversary* $\mathcal{A}$ *and for any non-negligible function* γ*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that the following probability is bounded by* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[D \text{ is } \gamma\text{-}\mathsf{Dist}^{\mathsf{pk},(\mathsf{sk},0)}_{\mathsf{params}} \text{ for } \mu \ : \ \begin{array}{c} (n,t) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow\$ \mathsf{Setup}(1^\lambda, 1^n, 1^t) \\ (D, \mu) \leftarrow \mathcal{A}(\mathsf{mpk}, \mathsf{pkc}, \{\mathsf{sk}_i\}_{i\in[n]}) \end{array}\right]$$

Definition 26 (Decoder-based Index Hiding Security). *A* TPLBE *scheme is said to satisfy decoder-based index hiding security if for any* PPT *adversary* $\mathcal{A}$ *and for any non-negligible function* γ*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all indices* $i^* \in [n]$*, the following probability is bounded by* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[D \text{ is } \gamma\text{-}\mathsf{Dist}^{i,i+1}_{\mathsf{params}} \text{ for } \mu \ : \ \begin{array}{c} (n,t) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow\$ \mathsf{Setup}(1^\lambda, 1^n, 1^t) \\ (D, \mu, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{mpk}, \mathsf{pkc}, \{\mathsf{sk}_i\}_{i\in[n]\setminus i^*}) \end{array}\right]$$

Definition 27 (Decoder-based Message Hiding Security). *A* TPLBE *scheme is said to satisfy decoder-based message hiding security if for any* PPT *adversary* $\mathcal{A}$ *and for any non-negligible function* γ*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that the following probability is bounded by* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[D \text{ is } \gamma\text{-}\mathsf{Dist}^{n}_{\mathsf{params}} \text{ for } \mu_0, \mu_1 \ : \ \begin{array}{c} (n,t) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow\$ \mathsf{Setup}(1^\lambda, 1^n, 1^t) \\ (D, \mu_0, \mu_1) \leftarrow \mathcal{A}(\mathsf{mpk}, \mathsf{pkc}, \{\mathsf{sk}_i\}_{i\in[n]}) \end{array}\right]$$

7.2 A generic compiler for TTT from Threshold PLBE

In this section, we show how to construct threshold traitor tracing scheme from threshold PLBE. Our compiler is based on the compiler from [46], and thus has two parts. First, we show that a threshold PLBE scheme that achieves 1-bounded security, also satisfies the decoder based security. Next, we show how to construct a threshold traitor tracing scheme from threshold PLBE scheme that achieves decoder-based security.

7.2.1 Decoder-based Threshold PLBE from 1-bounded Threshold PLBE.

Let TPLBE be a threshold PLBE scheme that satisfies 1-bounded security. Then, using arguments similar to [46, Lemma 4.1, Lemma 4.2, and Lemma 4.3], the same scheme also satisfies the decoder-based security.

Lemma 7. *If* TPLBE *satisfies 1-bounded indistinguishability (Definition 21), then it also satisfies decoder-based indistinguishability (Definition 25) property.*

Proof. Same as proof of Lemma 4.1 of [46].

Remark. In the security proof Lemma 4.1 of [46], all the secret keys are generated honestly, and moreover, the reduction works, even when the adversary gets access to all n decryption keys. Therefore, the threshold structure of the secret keys in TPLBE does not alter the security reduction compared to the non-threshold version.

Lemma 8. *If* TPLBE *satisfies 1-bounded index hiding (Definition 22), then it also satisfies decoder-based index-hiding (Definition 26) property.*

Proof. As in the proof of [46, Lemma 4.2], this proof is similar to the proof of Lemma 7, except that the reduction algorithm queries for either a secret key encryption of the message μ for index i or for index $i+1$ (depending upon $\beta \leftarrow\$ \{0,1\}$).

Lemma 9. *If* TPLBE *satisfies 1-bounded message hiding (Definition 23), then it also satisfies decoder-based message-hiding (Definition 27) property.*

Proof. As in the proof of [46, Lemma 4.3], this proof is similar to the proof of Lemma 7, except that the reduction algorithm queries for either a secret key encryption of the message μ_0 for index n or the encryption of μ_1 for index n (depending upon $\beta \leftarrow\$ \{0,1\}$).

7.2.2 TTT from Decoder-based Threshold PLBE. Let TPLBE be a threshold PLBE scheme with decoder-based security. We will use threshold PLBE to construct a TTT scheme $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Combine}, \mathsf{Trace})$. Our construction follows the same structure as [46], and we summarize it in Figure 11.

Correctness. This follows directly from the correctness of the threshold PLBE scheme.

Strong IND-CPA Security. Now, we will argue that given a threshold PLBE scheme TPLBE that is strong IND-CPA secure, our threshold traitor tracing scheme is also strong IND-CPA secure.

Theorem 9. *Assuming the TPLBE scheme* TPLBE *satisfies the security guarantee in Definition 20, the traitor tracing scheme is strong IND-CPA secure as per Definition 2.*

Proof. Suppose on the contrary, there exists an PPT adversary $\mathcal{A}_{\mathsf{TTT}}$, non-negligible function $\gamma(\cdot)$, such that $\mathcal{A}_{\mathsf{TTT}}$ breaks the strong IND-CPA security of the TTT scheme TTT. Then, we can use $\mathcal{A}_{\mathsf{TTT}}$ to build an adversary $\mathcal{A}_{\mathsf{TPLBE}}$, that will use $\mathcal{A}_{\mathsf{TTT}}$ to break the strong IND-CPA of the TPLBE scheme TPLBE. Let Ch be the TPLBE challenger. $\mathcal{A}_{\mathsf{TPLBE}}$ works as follows.

- $\mathcal{A}_{\mathsf{TPLBE}}$ forwards 1^λ to $\mathcal{A}_{\mathsf{TTT}}$ and receives (n, t, state). $\mathcal{A}_{\mathsf{TPLBE}}$ then forwards them to Ch.
- $\mathcal{A}_{\mathsf{TPLBE}}$ receives $\mathsf{state}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}$ from Ch, and forwards them to $\mathcal{A}_{\mathsf{TTT}}$ as $(\mathsf{state}, \mathsf{pk}, \mathsf{pkc}, \mathsf{tk}) := (\mathsf{state}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk})$.
- $\mathcal{A}_{\mathsf{TPLBE}}$ forwards the secret key and partial decryption query from $\mathcal{A}_{\mathsf{TTT}}$ to its own oracle.
- Similarly, $\mathcal{A}_{\mathsf{TPLBE}}$ forwards the challenge messages output by $\mathcal{A}_{\mathsf{TTT}}$ to Ch, and forwards the received response ct it receives from Ch to $\mathcal{A}_{\mathsf{TTT}}$.
- Finally, when $\mathcal{A}_{\mathsf{TTT}}$ outputs a bit b', $\mathcal{A}_{\mathsf{TPLBE}}$ outputs b' as its guess.

It is easy to see that $\mathcal{A}_{\mathsf{TPLBE}}$ perfectly simulates the view of the $\mathcal{A}_{\mathsf{TTT}}$. Therefore, if $\mathcal{A}_{\mathsf{TTT}}$ breaks the strong IND-CPA security of TTT as per Definition 2 with some non-negligible probability $\gamma(\lambda)$, $\mathcal{A}_{\mathsf{TPLBE}}$ breaks the strong IND-CPA security of TPLBE as per Definition 20 with probability $\gamma(\lambda)$.

$\underline{\mathsf{Setup}(1^\lambda, 1^n, 1^t) \to (\mathsf{pp}, \mathsf{pk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) :}$

1. Let $(\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow\!\$\ \mathsf{TPLBE.Setup}(1^\lambda, 1^n, 1^t)$
2. **output** $(\mathsf{pp}, \mathsf{pk} := \mathsf{mpk}, \mathsf{pkc}, \mathsf{tk} := \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]})$

$\underline{\mathsf{Enc}(\mathsf{mpk}, \mu) \to \mathsf{ct} :}$

1. **output** $\mathsf{TPLBE.EncPK}(\mathsf{mpk}, \mu)$.

$\underline{\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}_x) \to d_i :}$

1. **output** $\mathsf{TPLBE.Dec}(\mathsf{sk}_i, \mathsf{ct})$.

$\underline{\mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}, \mathcal{J}, \{d_j\}_{j\in\mathcal{J}}) \to \mu :}$

1. **output** $\mathsf{TPLBE.Combine}(\mathsf{pkc}, \mathsf{ct}, \mathcal{J}, \{d_j\}_{j\in\mathcal{J}})$.

$\underline{\mathsf{Trace}(\mathsf{pk}, \mathsf{tk}, 1^y, \mu_0, \mu_1) :}$

1. Let $\epsilon := 1/y$ and $W := \lambda \cdot (n \cdot y)^2$.
2. For $i = 0 \ldots n$
   (a) Let $\mathsf{count}_i := 0$.
   (b) For $j = 1$ to W:
      i. Sample $b_{i,j} \leftarrow \{0, 1\}$.
      ii. Let $\mathsf{ct}_{i,j} \leftarrow\!\$\ \mathsf{TPLBE.EncSK}(\mathsf{msk}, \mu_{b_{i,j}}, i)$.
      iii. If $D(\mathsf{ct}_{i,j}) = b_{i,j}$, update $\mathsf{count}_i := \mathsf{count}_i + 1$.
   (c) Let $\hat{p}_i := \mathsf{count}_i/W$.
3. Let $\mathcal{J} := \{i \mid \hat{p}_{i-1} - \hat{p}_i \geq \epsilon/4n\}_{i\in[n]}$.
4. **output** $\mathcal{J}$.

Fig. 11. Our compiler to construct a threshold traitor tracing from TPLBE.

Traceability. We now argue the scheme we describe in Figure 11 achieves traceability.

Theorem 10. *Assuming* TPLBE *is a secure threshold PLBE scheme satisfying the decoder-based index hiding, message-hiding and indistinguishability properties (Definition 26,Definition 27,Definition 25), the threshold traitor tracing scheme* TTT *we describe in Figure 11, is traceable as per Definition 3.*

Proof. The proof of this theorem follows using a proof identical to the proof of correct tracing of (non-threshold) PLBE scheme as given in [46, Section 4.2.2]. This is because, both in the traceability experiment of TPLBE (see Figure 4) and the (non-threshold) PLBE (see [46, Figure 3]), the adversary can query secret keys of an arbitrary number (up to n) of decryptors before outputting the decoder box. Hence, the threshold structure of the keys does not play a role.

8 Threshold PLBE from ABTE and mixed FE

In this section, we present a generic compiler to construct threshold PLBE from (i) a mixed FE scheme and (ii) an attribute-based Threshold encryption.

Rough Idea. We define a compiler to obtain a Threshold PLBE, given an MFE and a key-policy ABTE. Roughly speaking, our compiler is a thresholdized version of the one given by Goyal et al in [46], and works as follows:

– The ciphertext is an ABTE ciphertext with an attribute which is either a 'normal' MFE ciphertext encrypting the identity function or a 'special' MFE ciphertext encrypting the comparison function. In more detail, the ciphertexts generated in the public-key mode (via MFE.EncPK) have as attribute the normal MFE ciphertext. For encrypting a message in the secret-key mode with respect to an index j, the attribute is the MFE ciphertext encrypting the function $f_{>j}$; for any input i, this function outputs 1 only if $i > j$.
– The secret key of each party i is an ABTE secret key, with the policy as the MFE decryption circuit hard-coded with the i-th MFE secret key
– To decrypt, if the ciphertext attribute is a normal MFE ciphertext then each party can use their ABTE secret key to partially decrypt the ABTE ciphertext. Finally, t many partial decryptions can be combined to recover the message. However, if the ciphertext attribute is a 'special' MFE ciphertext, only users with a policy $(\mathsf{MFE.Dec}(\mathsf{MFE.sk}_i, \cdot))$ that is satisfied by the ciphertext attribute are able to partially decrypt. This means that, a ciphertext encrypted with respect to index j can only be decrypted by a set of t parties all with index $i > j$.

We formally describe the compiled construction for TPLBE in Fig. 12 and prove that our scheme satisfies the correctness (Def. 19) and security definitions Def. 21, Def. 22 and Def. 23 in the theorems below.

Theorem 11. *If the underlying* MFE *and* ABTE *schemes are correct as per definitions Def. 6 and Def. 9 resp., then our* TPLBE *scheme also satisfies correctness as per definition Def. 19.*

Proof. We show that our TPLBE scheme is correct both in the public key and secret key modes. In our scheme, a ciphertext for a message μ is generated as $\mathsf{ct}_x \leftarrow\!\$\ \mathsf{ABTE.Enc}(\mathsf{mpk}, \mu, x)$ where x is the ciphertext attribute. The secret key of each user i is $\mathsf{ABTE.sk}_i$ generated under the policy $C_i := \mathsf{MFE.Dec}(\mathsf{MFE.sk}_i, \cdot)$.

– In the public-key encryption mode, the ciphertext attribute x is a normal MFE ciphertext, computed using the public key mode of MFE as $x \leftarrow\!\$\ \mathsf{MFE.EncPK}(\mathsf{MFE.mpk})$. The function f that is encrypted in a normal MFE ciphertext is the always-accepting function. So from correctness of MFE, it follows that $\mathsf{MFE.Dec}(\mathsf{MFE.sk}_i, x) = f(i) = 1$ except with negligible probability. This implies that the ciphertext attribute x satisfies the key-policy associated with $\mathsf{ABTE.sk}_i$ for all $i \in [n]$. therefore, each party is able to decrypt ct_x using ABTE decryption under their key $\mathsf{ABTE.sk}_i$ to get $d_i \leftarrow \mathsf{ABTE.Dec}(\mathsf{ABTE.sk}_i, \mathsf{ct}_x)$ and by correctness of ABTE, a set of t many partial decryption can be combined to recover the message $\mu \leftarrow \mathsf{ABTE.Combine}(\mathsf{ct}_x, \{d_i\}_{i \in S})$.
– The ciphertext attribute x in the secret-key encryption mode is a MFE ciphertext generated as $x \leftarrow\!\$\ \mathsf{MFE.Enc}(\mathsf{MFE.msk}, f_{>j})$ for some index j, where $f_{>j}$ is the comparison function that outputs 1 on all inputs $i > j$ and 0 otherwise. From correctness of the MFE scheme, $\mathsf{MFE.Dec}(\mathsf{MFE.sk}_i, x) = 1$ iff $i > j$ and 0 for $i \leq j$. So the ciphertext attribute x satisfies the policy associated with $\mathsf{ABTE.sk}_i$ only for indices $i > j$. Consequently, only parties with indices $i > j$ are able to decrypt ct_x using ABTE decryption. By correctness of ABTE, a set of t many such partial decryptions d_i with $i > j$ can be combined to recover $\mu \leftarrow \mathsf{ABTE.Combine}(\mathsf{ct}_x, \{d_i\}_{i \in S})$.

Theorem 12. *Assuming that we have an* ABTE *scheme that satisfies selective IND-CPA security (Def. 10), our* TPLBE *scheme is IND-CPA secure as per definition Def. 20.*

Proof. Let $\mathcal{A}_{\mathsf{TPLBE}}$ be a PPT adversary for the TPLBE scheme controlling upto $t-1$ parties. We argue that if $\mathcal{A}_{\mathsf{TPLBE}}$ breaks the strong-IND-CPA security of our scheme with non-negligible

$\mathsf{Setup}(1^\lambda, 1^n, 1^t) \to (\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]})$:

1. Sample $(\mathsf{MFE.pp}, \mathsf{MFE.mpk}, \mathsf{MFE.msk}) \leftarrow\!\$\ \mathsf{MFE.Setup}(1^\lambda)$.
2. Sample the MFE secret key for user i as $\mathsf{MFE.sk}_i \leftarrow\!\$\ \mathsf{MFE.KeyGen}(\mathsf{MFE.msk}, i)$.
3. Set $C_i := \mathsf{MFE.Dec}(\mathsf{MFE.sk}_i, \cdot)$.
4. Sample $(\mathsf{ABTE.pp}, \mathsf{ABTE.mpk}, \mathsf{ABTE.pkc}, \mathsf{ABTE.msk}) \leftarrow\!\$\ \mathsf{ABTE.Setup}(1^\lambda, 1^n, 1^t, 1^z)$.
5. Sample $\mathsf{sk}_i \leftarrow\!\$\ \mathsf{ABTE.KeyGen}(\mathsf{ABTE.msk}, i, C_i)\ \forall\ i\ \in [n]$.
6. Set $\mathsf{mpk} := (\mathsf{MFE.mpk}, \mathsf{ABTE.mpk})$, $\mathsf{pkc} := \mathsf{ABTE.pkc}$ and $\mathsf{msk} = \mathsf{MFE.msk}$.
7. Output the tuple $(\mathsf{mpk}, \mathsf{pkc}, \mathsf{msk})$, the public parameters $\mathsf{pp} := (\mathsf{MFE.pp}, \mathsf{ABTE.pp})$ and the secret keys $\{\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n\}$.

$\mathsf{EncPK}(\mathsf{mpk}, \mu)$:

1. Parse mpk as $(\mathsf{MFE.mpk}, \mathsf{ABTE.mpk})$.
2. Compute $x := \mathsf{MFE.EncPK}(\mathsf{MFE.mpk})$.
3. Output $\mathsf{ct}_x \leftarrow\!\$\ \mathsf{ABTE.Enc}(\mathsf{ABTE.mpk}, \mu, x)$

$\mathsf{EncSK}(\mathsf{msk}, \mu, j)$:

1. Parse msk as MFE.msk.
2. Compute $x = \mathsf{MFE.EncSK}(\mathsf{MFE.msk}, \mathsf{comp}_j)$, where $\mathsf{comp}_j : [n] \to \{0, 1\}$, such that $\mathsf{comp}_j(i) = 1\ \forall\ i > j$ and 0 otherwise.
3. Output $\mathsf{ct}_x \leftarrow\!\$\ \mathsf{ABTE.Enc}(\mathsf{ABTE.mpk}, \mu, x)$.

$\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}_x)$:

1. Output $d_i \leftarrow \mathsf{ABTE.Dec}(\mathsf{sk}_i, \mathsf{ct}_x)$.

$\mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}_x, \{d_j\}_{j\in\mathcal{S}})$:

1. Parse pkc as ABTE.pkc.
2. Output $\mathsf{ABTE.Combine}(\mathsf{ABTE.pkc}, \mathsf{ct}_x, \{d_j\}_{j\in\mathcal{S}})$.

Fig. 12. Our compiler to construct a threshold TPLBE from Mixed functional encryption MFE and attribute-based threshold encryption ABTE.

probability, then we can construct an adversary $\mathcal{B}_{\mathsf{ABTE}}$ that breaks the strong-IND-CPA security of the underlying ABTE scheme, also with non-negligible probability.

Let Ch be the ABTE challenger.

- Ch sends the security parameter 1^λ to $\mathcal{B}_{\mathsf{ABTE}}$.
- $\mathcal{B}_{\mathsf{ABTE}}$ forwards 1^λ to $\mathcal{A}_{\mathsf{TPLBE}}$ and receives back (n, t, state).
- $\mathcal{B}_{\mathsf{ABTE}}$ runs $\mathsf{MFE.Setup}(1^\lambda)$ to get the MFE parameters $(\mathsf{MFE.pp}, \mathsf{MFE.mpk}, \mathsf{MFE.msk})$. It sets the challenge attribute to be a normal MFE ciphertext, that is, $x^* = \mathsf{MFE.EncPK}(\mathsf{MFE.mpk})$.
- $\mathcal{B}_{\mathsf{ABTE}}$ responds to its challenger with the challenge parameters $(n, t, z, x^*, \mathsf{state})$ where x^* is the challenge attribute and $z \in \mathbb{N}$ is the length of MFE ciphertexts.
- Ch sends the ABTE public parameters $(\mathsf{ABTE.pp}, \mathsf{ABTE.mpk}, \mathsf{ABTE.pkc})$ to $\mathcal{B}_{\mathsf{ABTE}}$.
- With the help of these parameters $\mathcal{B}_{\mathsf{ABTE}}$ partly runs the Setup for TPLBE and generates the public parameters $\mathsf{TPLBE.pp} := (\mathsf{MFE.pp}, \mathsf{ABTE.pp})$, $\mathsf{TPLBE.pkc} := \mathsf{ABTE.pkc}$, $\mathsf{TPLBE.mpk} := (\mathsf{MFE.mpk}, \mathsf{ABTE.mpk})$ and the master secret key $\mathsf{TPLBE.msk} := \mathsf{MFE.msk}$. $\mathcal{B}_{\mathsf{ABTE}}$ then sends these TPLBE parameters to $\mathcal{A}_{\mathsf{TPLBE}}$.
- Observe that, $\mathcal{A}_{\mathsf{TPLBE}}$ can make at most $t-1$ secret key queries and any number of valid decryption queries. On the other hand, $\mathcal{B}_{\mathsf{ABTE}}$ can also make at most $t-1$ secret key queries

for satisfying policies, any number of secret key queries for unsatisfying policies, and arbitrarily many valid decryption queries to its challenger Ch. Hence, $\mathcal{B}_{\mathsf{ABTE}}$ can easily respond to $\mathcal{A}_{\mathsf{TPLBE}}$'s oracle queries as follows:

- $\mathcal{B}_{\mathsf{ABTE}}$ simulates the oracle $\mathsf{sk}(\cdot)$ for $\mathcal{A}_{\mathsf{TPLBE}}$ as follows: Initialize a list $\mathcal{J} := \emptyset$. On query $\mathsf{sk}(i)$, update $\mathcal{J} = \mathcal{J} \cup i$ and query Ch on (i, C_i) where $C_i := \mathsf{MFE.Dec}(\mathsf{MFE.sk}_i, \cdot)$ and $\mathsf{MFE.sk}_i \leftarrow\$ \mathsf{MFE.KeyGen}(\mathsf{msk}, i)$. Forward Ch's response to $\mathcal{A}_{\mathsf{TPLBE}}$. Note that this policy is indeed satisfied by x^*, but that is alright since $\mathcal{A}_{\mathsf{TPLBE}}$ can only make $t-1$ such queries.
- $\mathcal{B}_{\mathsf{ABTE}}$ simulates the oracle $\mathsf{dec}(\cdot)$ for $\mathcal{A}_{\mathsf{TPLBE}}$ as follows: On query $\mathsf{dec}(\mu, r, i)$, parse r as (r_0, r_1) and query Ch on $\mathsf{dec}(\mu, r_1, x, C_i)$ where C_i is the same as above, and $x = \mathsf{MFE.EncPK}(\mathsf{MFE.mpk}; r_0)$.
- Eventually, $\mathcal{A}_{\mathsf{TPLBE}}$ returns challenge messages $(\mu_0, \mu_1, \mathsf{state})$. $\mathcal{B}_{\mathsf{ABTE}}$ forwards this to its challenger Ch.
- Receives a challenge ciphertext ct_b^* which it forwards to $\mathcal{A}_{\mathsf{TPLBE}}$.
- Finally, $\mathcal{A}_{\mathsf{TPLBE}}$ outputs a bit b'. $\mathcal{B}_{\mathsf{ABTE}}$ forwards this to Ch.

  Note that the TPLBE ciphertext is nothing but an ABTE ciphertext. With the ABTE challenge attribute being set to the normal MFE ciphertext, $\mathcal{B}_{\mathsf{ABTE}}$ perfectly simulates the TPLBE semantic security experiment (Figure 10) for $\mathcal{A}_{\mathsf{TPLBE}}$. As a result, $\mathcal{B}_{\mathsf{ABTE}}$ wins the semantic security game for ABTE with the same probability as $\mathcal{A}_{\mathsf{TPLBE}}$.

Theorem 13. *Our* TPLBE *scheme satisfies q-bounded indistinguishability as long as the underlying* MFE *scheme is q-bounded restricted accept indistinguishable.*

Proof. The proof of this theorem is a satraightforward generalization of the proof of [46, Lemma 6.1] to support (i) arbitrary $q = \mathsf{poly}(\lambda)$ and (ii) the threshold setting. In more detail, we construct an adversary $\mathcal{B}$ for q-bounded restricted accept indistinguishability, that uses an adversary $\mathcal{A}$ for q-bounded indistinguishability. It receives parameters n, t from $\mathcal{A}$, It sends the function comp_0 as its challenger function to its challenger, and receives MFE parameters along with the challenger ciphertext ct^* in return. $\mathcal{B}$ samples the ABTE parameters for n, t on its own. It then queries its challenger for q encryptions to the function comp_0, which it can use to respond to $\mathsf{EncSK}(0, \cdot)$ queries by $\mathcal{A}$. The rest of the reduction is the same as that for the proof of [46, Lemma 6.1]

At a high level, the reason why the threshold does not affect the proof is that, in the security argument, the adversary is allowed to learn all n ABTE decryption keys, meaning that it is capable of decrypting the challenge ciphertext. Hence, the threshold does not play any role here, and we get indistiguishability simply owing to the q-bounded accept indistinuishability of the MFE.

Theorem 14. *Our* TPLBE *scheme statisfies* 1*-bounded index hiding as long as the underlying* MFE *scheme is* 1*-bounded restricted function indistinguishable.*

Proof. Same as proof of lemma 6.2 in [46]. Here too the security boils down only to the 1-bounded restricted function indistinguishability property of MFE and the proof is not affected by the threshold. The adversary is allowed to learn all but one ABTE secret keys, regardless of the threshold.

Theorem 15. *Our* TPLBE *scheme statisfies q-bounded message hiding as long as the underlying* ABTE *achieves selective CPA security.*

Proof. Let $\mathcal{A}$ be the q-bounded message hiding adversary for TPLBE. We argue that if $\mathcal{A}$ breaks message-hiding of our threshold PLBE scheme with a non-negligible probability δ then we can construct an adversary $\mathcal{B}$ that breaks the semantic security of ABTE, also with a non-negligible probability.

- Ch sends the security parameter 1^λ to $\mathcal{B}_{\mathsf{ABTE}}$.
- $\mathcal{B}_{\mathsf{ABTE}}$ forwards 1^λ to $\mathcal{A}_{\mathsf{TPLBE}}$ and receives back the parameters (n, t, state).
- $\mathcal{B}_{\mathsf{ABTE}}$ runs $\mathsf{MFE.Setup}(1^\lambda)$ to get the mixed-FE parameters $(\mathsf{MFE.pp}, \mathsf{MFE.mpk}, \mathsf{MFE.msk})$. It sets the challenge attribute to be a special MFE ciphertext $x^* := \mathsf{MFE.EncSK}(\mathsf{MFE.msk}, \mathsf{comp}_n)$.
- $\mathcal{B}_{\mathsf{ABTE}}$ responds to its challenger with the challenge parameters $(n, t, z, x^*, \mathsf{state})$ where x^* is the challenge attribute and $z \in \mathbb{N}$ is the length of MFE ciphertexts.
- Ch sends the ABTE public parameters $(\mathsf{ABTE.pp}, \mathsf{ABTE.mpk}, \mathsf{ABTE.pkc})$ to $\mathcal{B}_{\mathsf{ABTE}}$.
- Recall that $\mathcal{B}_{\mathsf{ABTE}}$ can submit any number of secret key queries for unsatisfying policies C (i.e. $C(x^*) = 0$). Additionally, it can query at most t-1 secret keys for satisfying policies.
- $\mathcal{B}_{\mathsf{ABTE}}$ submits secret key queries (i, C_i) with $C_i := \mathsf{MFE.Dec}(\mathsf{MFE.sk}_i, \cdot)$, where $\mathsf{MFE.sk}_i := \mathsf{MFE.KeyGen}(\mathsf{msk}, i)$ for all $i \in [n]$. This is allowed since $C_i(x^*) = 0$ for all $i \in [n]$. $\mathcal{B}$ gets back ABTE secret keys $\mathsf{ABTE.sk}_i$ for all n parties from its challenger.
- $\mathcal{B}$ sets $\mathsf{TPLBE.pp} := (\mathsf{MFE.pp}, \mathsf{ABTE.pp})$, $\mathsf{TPLBE.pkc} := \mathsf{ABTE.pkc}$, $\mathsf{TPLBE.mpk} := (\mathsf{MFE.mpk}, \mathsf{ABTE.mpk})$ and the master secret key $\mathsf{TPLBE.msk} := \mathsf{MFE.msk}$ as well as the individual secret keys $\mathsf{TPLBE.sk}_i := \mathsf{ABTE.sk}_i$ for all $i \in [n]$. $\mathcal{B}_{\mathsf{ABTE}}$ then sends these TPLBE parameters to $\mathcal{A}_{\mathsf{TPLBE}}$.
- $\mathcal{A}_{\mathsf{TPLBE}}$ can submit up to q many secret key encryption queries. Suppose it submits a query to $\mathsf{TPLBE.EncSK}(\mathsf{msk}, .)$ for message μ and index i. $\mathcal{B}_{\mathsf{ABTE}}$ responds to such queries honestly. It can do so because it knows the MFE master secret key.
- $\mathcal{A}_{\mathsf{TPLBE}}$ submits two challenge messages μ_0, μ_1 to $\mathcal{B}_{\mathsf{ABTE}}$. $\mathcal{B}_{\mathsf{ABTE}}$ forwards them to its challenger and receives back a challenge ciphertext ct_b which it sends to $\mathcal{A}_{\mathsf{TPLBE}}$.
- $\mathcal{B}_{\mathsf{ABTE}}$ forwards the guess b' output by $\mathcal{A}_{\mathsf{TPLBE}}$ to Ch.

Clearly, $\mathcal{B}$ perfectly simulates the message-hiding game to $\mathcal{A}$. The theorem then follows from thet fact that if $\mathcal{A}$ wins its game, then $\mathcal{B}$ also trivially wins its game.

9 Risky and Limited Threshold Traitor Tracing from Weak Threshold PLBE

In this section, we present our generic compiler to construct a Risky and Limited TTT scheme from a weak threshold PLBE. We start with defining weak threshold PLBE, before describing our compiler.

Definition 28 (Weak Decoder-based Indistinguishability). *A* TPLBE *scheme is said to satisfy weak selective decoder-based indistinguishability if there are constants $\alpha, \beta, \gamma \in (0, 1)$ such that, for every polynomial $\kappa(\lambda)$ and every stateful* PPT *adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)(\lambda)$ such that for every λ, the following is true:*

- *Run $\mathcal{A}(1^\lambda)$ to get n, t.*
- *Now run $(\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow\$ \mathsf{Setup}(1^\lambda, 1^n, 1^t)$ and send $\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \{\mathsf{sk}_i\}_{i\in[n]}$ to $\mathcal{A}$.*
- *$\mathcal{A}$ outputs a decoder D and messages μ_0, μ_1.*
- *Let $\mathsf{Adv}(D, \mathsf{mpk}) = 2\Pr[D(\mathsf{ct}) = b : \substack{b\leftarrow\$\{0,1\} \\ \mathsf{ct}\leftarrow\$\mathsf{EncPK}(\mathsf{mpk},\mu_b)}] - 1$ be the advantage of D in decrypting for publicly generated ciphertexts. Let $\mathsf{Adv}'(D, \mathsf{msk}) = 2\Pr[D(\mathsf{ct}) = b : \substack{b\leftarrow\$\{0,1\} \\ \mathsf{ct}\leftarrow\$\mathsf{EncSK}(\mathsf{msk},\mu_b,0)}] - 1$ be the advantage for secret-key ciphertexts for index 0.*
  Let $\mathsf{Good}_\alpha(D, \mathsf{mpk})$ be the event $\mathsf{Adv}(D, \mathsf{mpk}) \geq \alpha$, and let $\mathsf{Good}'_\beta(D, \mathsf{msk})$ be the event $\mathsf{Adv}'(D, \mathsf{msk}) \geq \beta$. Then we have $\Pr[\mathsf{Good}'_\beta(D, \mathsf{msk})] \geq \gamma \Pr[\mathsf{Good}_\alpha(D, \mathsf{mpk})] - \mathsf{negl}(\lambda)$.

Definition 29 (Weak Threshold PLBE). *A* TPLBE *scheme is said to be a (α, β, γ)-weak TPLBE if it satisfies (α, β, γ)-weak Decoder-based Indistinguishability along with strong CPA security and decoder-based index and message hiding, as defined in Section 7.1.*

9.1 Achieving Weak Decoder-based Indistinguishability from 0-bounded Indistinguishability

In this section, we show how we can compile a TPLBE with 0-bounded indistinguishability to a new TPLBE construction that achieves weak decoder-based indistinguishability, while maintaining all the other security properties.

$\underline{\mathsf{Setup}(1^\lambda, 1^n, 1^t)}$:

1. Fix any constants $\ell, \alpha, \beta, \gamma$ such that
$$0 < \beta < \alpha < 1 \quad , \quad 0 < \gamma < 1 \quad , \quad \ell \in \mathbb{N} \quad \text{and} \quad \ell(1-\gamma)(\alpha-\beta)^2 > 4 \ .$$
2. For $\zeta \in [\ell]$, sample $(\mathsf{pp}_\zeta, \mathsf{mpk}_\zeta, \mathsf{pkc}_\zeta, \mathsf{msk}_\zeta, \{\mathsf{sk}_{\zeta,i}\}_{i\in[n]}) \leftarrow\!\!\$\ \mathsf{TPLBE.Setup}(1^\lambda, 1^n, 1^t)$.
3. Output $\mathsf{pp} := \{\mathsf{pp}_\zeta\}_{\zeta\in[\ell]}$, $\mathsf{mpk} := \{\mathsf{mpk}_\zeta\}_{\zeta\in[\ell]}$, $\mathsf{pkc} := \{\mathsf{pkc}_\zeta\}_{\zeta\in[\ell]}$, $\mathsf{msk} := \{\mathsf{msk}_\zeta\}_{\zeta\in[\ell]}$, and $\mathsf{sk}_i := \{\mathsf{sk}_{\zeta,i}\}_{\zeta\in[\ell]}$ for all $i \in [n]$.

$\underline{\mathsf{EncPK}(\mathsf{mpk}, \mu)}$:

1. Sample $\zeta \leftarrow\!\!\$\ [\ell]$, compute $\hat{\mathsf{ct}} \leftarrow\!\!\$\ \mathsf{TPLBE.EncPK}(\mathsf{mpk}_\zeta, \mu)$. Output $\mathsf{ct} := (\zeta, \hat{\mathsf{ct}})$.

$\underline{\mathsf{EncSK}(\mathsf{msk}, \mu, j)}$:

1. Sample $\zeta \leftarrow\!\!\$\ [\ell]$, compute $\hat{\mathsf{ct}} \leftarrow\!\!\$\ \mathsf{TPLBE.EncSK}(\mathsf{msk}_\zeta, \mu, j)$. Output $\mathsf{ct} := (\zeta, \hat{\mathsf{ct}})$.

$\underline{\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})}$:

1. Parse ct as $(\zeta, \hat{\mathsf{ct}})$, and sk_i as $\{\mathsf{sk}_{1,i}, \ldots, \mathsf{sk}_{\ell,i}\}$. Output $\mathsf{TPLBE.Dec}(\mathsf{sk}_{\zeta,i}, \hat{\mathsf{ct}})$.

$\underline{\mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}, \{d_j\}_{j\in S})}$:

1. Parse ct as $(\zeta, \hat{\mathsf{ct}})$. Output $\mathsf{TPLBE.Combine}(\mathsf{pkc}_\zeta, \hat{\mathsf{ct}}, \{d_j\}_{j\in S})$.

Fig. 13. Our compiler for constructing a TPLBE scheme TPLBE' with weak Decoder-based Indistinguishability from TPLBE with only 0-bounded Indistinguishability. Note that our compiler preserves all the other properties, i.e. strong CPA security, decoder-based index hiding and message hiding.

Correctness. Correctness of the resulting scheme TPLBE' follows trivially from correctness of the underlying scheme TPLBE.

Strong CPA security. Theorem 16 below proves that the scheme TPLBE' satisfies strong CPA security if TPLBE also satisfies strong CPA security.

Theorem 16. *For any* PPT *adversary* $\mathcal{A}$*, there exists another* PPT *adversary* $\mathcal{B}$ *such that,*

$$\mathsf{Adv}^{\mathsf{strong-cpa}}_{\mathcal{A},\mathsf{TPLBE}'}(\lambda) \leq \mathsf{Adv}^{\mathsf{strong-cpa}}_{\mathcal{B},\mathsf{TPLBE}}(\lambda)$$

Proof. The proof essentially follows the proof of Theorem 19. $\mathcal{B}$ samples an index $j^* \leftarrow\!\!\$\ [\ell]$, and plants the challenge TPLBE parameters at the j^*th index, and samples the TPLBE parameters for all other indices on its own. $\mathcal{B}$ simply forwards all oracle queries of $\mathcal{A}$ to its challenger. Eventually, $\mathcal{A}$ outputs the messages which it forwards to its challenger. $\mathcal{B}$ then uses j^* as the index for the challenge ciphertext, and simply forwards $\mathcal{A}$'s response bit as its response. It is easy to see that $\mathcal{B}$ wins its strong CPA game if $\mathcal{A}$ wins its strong CPA game. This proves the theorem.

Weak Decoder-based Indistinguishability. We prove weak Decoder-based Indistinguishability of TPLBE' in the theorem below.

Theorem 17. *If* TPLBE *achieves* 0*-bounded indistinguishability, then* TPLBE' *achieves weak decoder-based Indistinguishability.*

Proof. The proof is essentially identical to the proof of [68, Theorem 44], except for the following (trivial) changes:

- The term p_ζ for all $\zeta \in [\ell]$ is defined with respect to secret-key encryptions to 0, i.e. $\mathsf{EncSK}(\mathsf{msk}, \mu_b, 0)$ (where $b \leftarrow\!\$\ \{0,1\}$ is sampled randomly).
- Since we are considering an encryption scheme instead of a key encapsulation mechanism, the decoder in the 0-bounded indistinguishability game is accompanied with a message μ for which it can distinguish between public and secret-key ciphertexts.

Decoder-based Index Hiding. As in [68], this follows immediately from decoder-based index hiding of the underlying scheme TPLBE.

Decoder-based Message Hiding. As in [68], this follows immediately from decoder-based message hiding of the underlying scheme TPLBE.

9.2 Constructing Risky and Limited TTT from Weak TPLBE

We now present our compiler to construct Risky and Limited TTT from a weak TPLBE scheme. The compiler is essentially the same as that given in Section 7.2, but we can only achieve Risky and Limited tracing because of weak decoder-based Indistinguishability of the TPLBE scheme.

Correctness and strong CPA security of our compiled scheme follow directly from correctness and strong CPA of the TPLBE scheme. Since the compiler is the same as in Section 7.2, we refer the reader to the corresponding proofs in Section 7.2.

Risky and Limited Traceability. We prove that the resulting scheme achieves risky and Limited traceability in the theorem below.

Theorem 18. *If* TPLBE *is a* (α, β, γ)*-weak threshold PLBE (as in Definition 29), then the compiled scheme achieves* $(\alpha/2, \gamma)$*-risky Limited traceability.*

Proof. The proof is essentially identical to the proof of [68, Theorem 48]. Hence, we just provide a sketch here. First, let us use p_i to denote the probability with which D decrypts secret-key ciphertexts encrypted to $i \in \{0, \ldots, n\}$. To see why honest users are never accused, observe that by decoder based index hiding security of TPLBE, an adversary without the key of a party i cannot distinguish between ciphertexts encrypted to index $i-1$ and i, i.e. p_i and p_{i-1} must be negligibly close. Then, by chernoff bounds, we know that $\hat{p}_i$ is negligibly close to p_i for all i. Combining the two, we get that $\hat{p}_{i-1} - \hat{p}_i$ will be less than $\beta/8n$ except with negligible probability.

We now show that if the decoder is sufficiently good, some user will be accused with reasonable probability. For a decoder D and public key mpk, recall that $\mathsf{Good}_\alpha(D, \mathsf{mpk})$ means $2 \times \Pr\left[D(\mathsf{ct}) = b : \substack{b \leftarrow\!\$\{0,1\} \\ \mathsf{ct} \leftarrow\!\$ \mathsf{EncPK}(\mathsf{mpk}, \mu_b)}\right] - 1 \geq \alpha$. Similarly, $\mathsf{Good}'_\beta(D, \mathsf{msk})$ means

$$2 \times \Pr\left[D(\mathsf{ct}) = b : \substack{b \leftarrow\!\$\{0,1\} \\ \mathsf{ct} \leftarrow\!\$ \mathsf{EncSK}(\mathsf{msk}, \mu_b, 0)}\right] - 1 \geq \beta.$$

$\underline{\mathsf{KeyGen}(1^\lambda, 1^n, 1^t)}$:

1. Sample $(\mathsf{pp}, \mathsf{mpk}, \mathsf{pkc}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow\!\!\$\ \mathsf{TPLBE.Setup}(1^\lambda, 1^n, 1^t)$, and output $(\mathsf{pp}, \mathsf{pk} := \mathsf{mpk}, \mathsf{pkc}, \mathsf{tk} := \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]})$.

$\underline{\mathsf{Enc}(\mathsf{pk}, \mu)}$:

1. Output $\mathsf{TPLBE.EncPK}(\mathsf{mpk}, \mu)$.

$\underline{\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})}$:

1. Output $\mathsf{TPLBE.Dec}(\mathsf{sk}_i, \mathsf{ct})$.

$\underline{\mathsf{Combine}(\mathsf{pkc}, \mathsf{ct}, \{d_j\}_{j\in\mathcal{S}})}$:

1. Output $\mathsf{TPLBE.Combine}(\mathsf{pkc}, \mathsf{ct}, \{d_j\}_{j\in\mathcal{S}})$.

$\underline{\mathsf{Trace}^{D(\cdot)}(\mathsf{pk}, \mathsf{tk}, \mu_0, \mu_1)}$:

1. Let $W = \lambda \cdot (2n/\beta)^2$. Parse tk as msk.
2. For $i = 0 \ldots n$
   (a) Let $\mathsf{count}_i := 0$.
   (b) For $j = 1$ to W:
      i. Sample $b_{i,j} \leftarrow \{0, 1\}$.
      ii. Let $\mathsf{ct}_{i,j} \leftarrow\!\!\$\ \mathsf{TPLBE.EncSK}(\mathsf{msk}, \mu_{b_{i,j}}, i)$.
      iii. If $D(\mathsf{ct}_{i,j}) = b_{i,j}$, update $\mathsf{count}_i := \mathsf{count}_i + 1$.
   (c) Let $\hat{p}_i := \mathsf{count}_i/W$.
3. Let $\mathcal{J} := \{i \mid \hat{p}_{i-1} - \hat{p}_i \geq \beta/8n\}_{i\in[n]}$.
4. **output** $\mathcal{J}$.

Fig. 14. Our compiler to construct a $(\epsilon := \alpha/2, \delta := \gamma)$-risky Limited TTT from a (α, β, γ)-weak threshold PLBE scheme TPLBE. Note the factor of two in ϵ is just an artifact of the slight difference in definitions between TTT and weak decoder indistinguishability as defined in [68].

First, observe that by decoder-based message-hiding of TPLBE, we know that p_n must be $1/2 + \mathsf{negl}(\lambda)$ because $\mathsf{comp}_n(j) = 0$ for all parties $j \in [n]$. Next, if $\mathsf{Good}'^{D}_{\beta}(\mathsf{msk})$ occurs, we know that the box can decrypt messages encrypted to index 0, i.e. $p_0 = 1/2 + \beta/2$. Hence, there must exist a index i^* such that $p_{i^*-1} - p_{i^*} > \beta/2n$. Combining this with Chernoff bound, we get that $\hat{p}_{i^*-1} - \hat{p}_{i^*} \geq \beta/8n$ except with negligible probability. Hence, the tracer will accuse party i^*, in the event $\mathsf{Good}'^{D}_{\beta}(\mathsf{msk})$. By (α, β, γ)-weak decoder-based indistinguishability, we know,

$$\Pr[\mathsf{GoodTr}] = \Pr[\mathsf{Good}'^{D}_{\beta}(\mathsf{msk})] \geq \gamma \Pr[\mathsf{Good}^{D}_{\alpha}(\mathsf{mpk})] - \mathsf{negl}(\lambda) \ .$$

By observing that the probability of $\mathsf{GoodDec}$ is equal to that of $\mathsf{Good}^{D}_{\alpha}(\mathsf{mpk})$ by definition, we get the theorem.

10 Eliminating Riskiness and Limitedness in Tracing

10.1 Risk Mitigation Compiler

Figure 15 presents our compiler for eliminating risk. This compiler generalizes the risk mitigation compiler given in [67] to the threshold setting. Note that the Trace algorithm is not given ϵ (the

$\underline{\mathsf{KeyGen}(1^\lambda, n, t)}$:

1. For $j \in [T]$, sample // $T = (\beta/\alpha) \cdot \omega(\log \lambda)$

$$(\mathsf{pk}_j, \mathsf{pkc}_j, \mathsf{sk}_{j,1}, \ldots, \mathsf{sk}_{j,n}, \mathsf{tk}_j) \leftarrow\!\$\ \mathsf{TTT}_{\mathsf{risky,Limited}}.\mathsf{KeyGen}(1^\lambda, n, t)$$

2. Output $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_T)$, $\mathsf{pkc} = (\mathsf{pkc}_1, \ldots, \mathsf{pkc}_T)$, $\mathsf{tk} = (\mathsf{tk}_1, \ldots, \mathsf{tk}_T)$ and $\mathsf{sk}_i = (\mathsf{sk}_{1,i}, \ldots, \mathsf{sk}_{T,i})$ for all $i \in [n]$.

$\underline{\mathsf{Enc}(\mathsf{pk}, \mu)}$:

1. Sample $j \leftarrow\!\$\ [T]$. Output $(j, \mathsf{TTT}_{\mathsf{risky,Limited}}.\mathsf{Enc}(\mathsf{pk}_j, \mu))$.

$\underline{\mathsf{Dec}(\mathsf{sk}_i, (j, c))}$:

1. Output $d_i = \mathsf{TTT}_{\mathsf{risky,Limited}}.\mathsf{Dec}(\mathsf{sk}_{j,i}, c)$.

$\underline{\mathsf{Combine}(\mathsf{pkc}, (j, c), \{d_i\}_{i \in \mathcal{S}})}$:

1. Output $\mathsf{TTT}_{\mathsf{risky,Limited}}.\mathsf{Combine}(\mathsf{pkc}_j, c, \{d_i\}_{i \in \mathcal{S}})$.

$\underline{\mathsf{Trace}^{D(\cdot)}(\mathsf{pk}, \mathsf{tk}, \mu_0, \mu_1)}$:

1. For $j \in [T]$, let D_j be the decoder $D_j(c, \mu_0, \mu_1) = D((j, c), \mu_0, \mu_1)$.
2. For $j \in [T]$, run $\mathcal{J}_j^* \leftarrow\!\$\ \mathsf{TTT}_{\mathsf{risky,Limited}}.\mathsf{Trace}^{D_j(\cdot)}(\mathsf{pk}_j, \mathsf{tk}_j, \mu_0, \mu_1)$.
3. Output $\mathcal{J}^* = \bigcup_{j \in [T]} \mathcal{J}_j^*$.

Fig. 15. A generic compiler to get a β-risky Limited Threshold traitor tracing scheme $\mathsf{TTT}_{\mathsf{Limited}}$ from an α-risky Limited threshold traitor tracing scheme $\mathsf{TTT}_{\mathsf{risky,Limited}}$. Setting $\beta = 1$ gives us a Limited Threshold traitor tracing scheme.

probability with which a decoder box 'works') as input since this is a Limited scheme, meaning that ϵ is implicit in the public parameters of the scheme.

Correctness. Correctness of this compiler follows trivially from correctness of the underlying scheme $\mathsf{TTT}_{\mathsf{risky,Limited}}$. Specifically, for any $j \in [T]$, an honestly generated ciphertext with respect to the jth keys will be correctly decrypted by any set of t parties. Moreover, observe that the size of public and secret keys only grows by a factor of T, which is independent of the number of parties.

Strong CPA security. Theorem 19 below proves Strong-CPA security of the compiled scheme $\mathsf{TTT}_{\mathsf{Limited}}$ by relying on Strong-CPA security of the underlying scheme $\mathsf{TTT}_{\mathsf{risky,Limited}}$.

Theorem 19. *For any* PPT *adversary* $\mathcal{A}$*, there exists another* PPT *adversary* $\mathcal{B}$ *such that,*

$$\mathsf{Adv}^{\mathrm{strong-cpa}}_{\mathcal{A},\mathsf{TTT}_{\mathsf{Limited}}}(\lambda) \leq \mathsf{Adv}^{\mathrm{strong-cpa}}_{\mathcal{B},\mathsf{TTT}_{\mathsf{risky,Limited}}}(\lambda).$$

Proof. We will construct $\mathcal{B}$ that breaks Strong CPA for the underlying scheme, using $\mathcal{A}$. The game starts with $\mathcal{A}$ sending n, t. $\mathcal{B}$ simply forwards these to the Strong CPA challenger for $\mathsf{TTT}_{\mathsf{risky,Limited}}$. $\mathcal{B}$ gets back $\mathsf{pk}^*, \mathsf{pkc}^*, \mathsf{tk}^*$ from its challenger. It then samples $j^* \leftarrow\!\$\ [T]$ (this is the index that the challenge ciphertext will be encrypted to), and sets $\mathsf{pk}_{j^*} = \mathsf{pk}^*, \mathsf{pkc}_{j^*} = \mathsf{pkc}^*$ and $\mathsf{tk}_{j^*} = \mathsf{tk}^*$. $\mathcal{B}$ honestly samples the keys for all other indices. Formally, $\mathcal{B}$ samples $(\mathsf{pk}_j, \mathsf{pkc}_j, (\mathsf{sk}j, 1, \ldots, \mathsf{sk}_{j,n}), \mathsf{tk}_j) \leftarrow\!\$\ \mathsf{TTT}_{\mathsf{risky,Limited}}.\mathsf{KeyGen}(1^\lambda, n, t)$ for all $j \in [T] \backslash \{j^*\}$. $\mathcal{B}$ forwards $\{\mathsf{pk}_j\}_{j \in [T]}, \{\mathsf{pkc}_j\}_{j \in [T]}$ and $\{\mathsf{tk}_j\}_{j \in [T]}$ to $\mathcal{A}$. We now describe how it responds to oracle queries:

– $\mathsf{sk}(i)$. $\mathcal{B}$ queries its challenger for $\mathsf{sk}(i)$, let us denote the response with sk^*. It sets $\mathsf{sk}_{j^*,i} = \mathsf{sk}^*$ and responds with $\{\mathsf{sk}_{j,i}\}_{j\in[n]}$.
– $\mathsf{dec}(\mu, r, i)$. Let us parse the randomness r as $(j, \hat{r})$ where $j \in [T]$ denotes the random index to encrypt to. If $j \neq j^*$, then $\mathcal{B}$ simply responds with $\mathsf{TTT}_{\mathsf{risky,Limited}}.\mathsf{Dec}(\mathsf{sk}_{j,i}, \mathsf{TTT}_{\mathsf{risky,Limited}}.\mathsf{Enc}(\mathsf{pk}_j, \mu; \hat{r}))$. Otherwise, it queries its challenger for $\mathsf{dec}(\mu, \hat{r}, i)$, and simply forwards the response to $\mathcal{A}$.

Finally, $\mathcal{B}$ receives messages μ_0, μ_1 from $\mathcal{A}$, which it forwards to its challenger. Let us denote the response by c^*. $\mathcal{B}$ then forwards j^*, c^* to $\mathcal{A}$ as the challenge ciphertext. $\mathcal{B}$ then forwards $\mathcal{A}$'s guess b' to its challenger.

It can be seen that $\mathcal{B}$ perfectly simulates the Strong CPA game to $\mathcal{A}$. The challenge ciphertext is also distributed identically, since j^* is sampled randomly from $[T]$. Hence, if $\mathcal{A}$ can break Strong-CPA with non-negligible advantage, then $\mathcal{B}$ breaks Strong-CPA for the underlying scheme with the same advantage.

Traceability. Theorem 20 below proves that the compiler results in a β-risky Limitedscheme. This implies that by setting $\beta = 1$, we can indeed eliminate risk via this compiler.

Theorem 20. *Assume $T = (\beta/\alpha) \cdot \omega(\log \lambda)$. If $\mathsf{TTT}_{\mathsf{risky,Limited}}$ is an α-risky Limited threshold traitor tracing scheme, then $\mathsf{TTT}_{\mathsf{Limited}}$ is a β-risky Limited threshold traitor tracing scheme.*

Proof (Sketch). The proof is the same as the proof for Theorem 10 in [67]. Roughly speaking, since the adversary in the tracing game is allowed to access the secret keys of all n parties, the threshold t does not matter. Hence, we only give a proof sketch below, and refer the reader to [67] for the full proof.

Consider a decoder D^* outputted by an adversary. We say that $j \in [T]$ is "good" if D^*_j has a high chance of decrypting ciphertexts for index j of $\mathsf{TTT}_{\mathsf{risky,Limited}}$. D^* can only decrypt ciphertexts for j where D^*_j is good; thus the event $\mathsf{GoodDec}$ (for the compiled scheme $\mathsf{TTT}_{\mathsf{Limited}}$) implies that the fraction of good j is large. Since each j represents a different instance of the risky Limited scheme, each of the decoders D^*_j should intuitively have an α chance of being traced to some user. As long as the number of good j is larger than $\omega(\log \lambda)/\alpha$, then we would expect that, with overwhelming probability, at least one of the D^*_j traces. One challenge is that, for general decoders with arbitrary inverse-polynomial success probability, the number of good j may be small. In particular, if D^* has advantage $1/T$, then it could be that only a single j is good. We resolve this by only guaranteeing *Limited* security, which implies that a constant fraction of j are good. Another challenge is that the attacker can choose adaptively which of the j will good and hence traceable, so the tracing probabilities are not independent events. Nevertheless, by a careful analysis, we are able to prove security.

10.2 Limitedness Mitigation Compiler

Figures 16 and 17 present our compiler for eliminating Limitedness. This compiler generalizes the compiler given in [67] in two ways: (i) we consider the t-out-of-n threshold setting, (ii) we construct an encryption scheme instead of a KEM, by using the standard hybrid encryption approach.

We start with describing the Goldreich-Levin Decoding algorithm which forms a crucial part of the compiler.

Theorem 21 (Goldreich-Levin Decoding [43]). *There exists a constant Γ and oracle algorithm $\mathsf{GL}^D(\ell, \epsilon')$ running in time $\mathsf{poly}(\ell, \log(1/\epsilon'))$ and making $\mathsf{poly}(\ell, \log(1/\epsilon'))$ queries to D, such that the following holds. If there exists an $x \in \{0,1\}^\ell$ such that $\Pr[D(r) = x \cdot r \bmod 2 : r \leftarrow \{0,1\}^\ell] \geq 1/2 + \epsilon'$, then $\Pr[\mathsf{GL}^D(\ell, \epsilon') = x] \geq \Gamma \times (\epsilon')^2$.*

$\underline{\mathsf{KeyGen}(1^\lambda, n, t)}$:

1. Run $(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{tk}) \leftarrow\!\$\ \mathsf{TTT}_{\mathsf{Limited}}.\mathsf{KeyGen}(1^\lambda, n, t)$. Output $(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{tk})$.

$\underline{\mathsf{Enc}(\mathsf{pk}, \mu)}$:

1. Let $y = \omega(\log \lambda)$ and $z = z(\lambda)$ be a polynomial. // $\mu \in \{0,1\}^z$
2. For all $u \in [y], v \in [z]$, sample $\mu_{u,v} \leftarrow\!\$\ \mathcal{M}$ and compute $c_{u,v} \leftarrow\!\$\ \mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Enc}(\mathsf{pk}, \mu_{u,v})$.
3. For all $v \in [z]$, compute $\mu_v := \mu_{1,v} \oplus \mu_{2,v} \ldots \oplus \mu_{y,v}$.
4. Sample $s \leftarrow\!\$\ \mathcal{M}$, and compute $b_v := \langle s, \mu_v \rangle \bmod 2$ for all $v \in [z]$. // $\langle s, \mu \rangle$ denotes bit-wise inner-product
5. Define $k := (b_1, \ldots, b_z) \in \{0,1\}^z$. Output $c = (\mu \oplus k, s, \{c_{u,v}\}_{u \in [y], v \in [z]})$.

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

1. Parse c as $(\hat{c}, s, \{c_{u,v}\}_{u \in [y], v \in [z]})$.
2. For all $u \in [y], v \in [z]$, compute $d_{u,v,i} := \mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Dec}(\mathsf{sk}_i, c_{u,v})$.
3. Output $d_i := \{d_{u,v,i}\}_{u \in [y], v \in [z]}$.

$\underline{\mathsf{Combine}(\mathsf{pkc}, c, \{d_j\}_{j \in \mathcal{S}})}$:

1. Parse c as $(\hat{c}, s, \{c_{u,v}\}_{u \in [y], v \in [z]})$ and d_i as $\{d_{u,v,i}\}_{u \in [y], v \in [z]}$ for all $i \in \mathcal{S}$.
2. For all $u \in [y], v \in [z]$, compute $\mu'_{u,v} \leftarrow \mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Combine}(\mathsf{pkc}, c_{u,v}, \{d_{u,v,i}\}_{i \in \mathcal{S}})$.
3. For all $v \in [z]$, compute $\mu'_v := \mu'_{1,v} \oplus \ldots \oplus \mu'_{y,v}$, and $b'_v := \langle s, \mu'_v \rangle \bmod 2$.
4. Define $k' := (b'_1, \ldots, b'_z) \in \{0,1\}^z$, and output $\mu' := \hat{c} \oplus k'$.

Fig. 16. A generic compiler to get a Threshold traitor tracing scheme TTT from a limited threshold traitor tracing scheme $\mathsf{TTT}_{\mathsf{Limited}}$. Here the message space for $\mathsf{TTT}_{\mathsf{Limited}}$ is assumed to be $\mathcal{M} := \{0,1\}^\ell$. The Trace algorithm is described in Fig. 17.

Correctness. Correctness follows almost directly from correctness of the underlying scheme $\mathsf{TTT}_{\mathsf{Limited}}$. In more detail, for all $u \in [y], v \in [z]$, $\mu'_{u,v} = \mu_{u,v}$ with overwhelming probability, by correctness of $\mathsf{TTT}_{\mathsf{Limited}}$. This means that, for all $v \in [z]$, $\mu'_v = \mu_v$, and, $b'_v = b_v$, which implies $k' = k$. Hence, $\mu' = (\mu \oplus k) \oplus k' = \mu$.

Also note that the public key and secret keys of the resulting scheme are of the same size as the original scheme. Moreover, the ciphertext size grows by a factor of $y \cdot z$ which is $\mathsf{poly}(\lambda)$.

Strong CPA Security. Theorem 22 below proves Strong CPA security of the compiled scheme by relying on strong CPA security of the underlying Limited scheme.

Theorem 22. *For any PPT adversary $\mathcal{A}$, there exists another PPT adversary $\mathcal{B}$ such that,*

$$\mathsf{Adv}^{\mathrm{strong-cpa}}_{\mathcal{A},\mathsf{TTT}}(\lambda) \leq 4 \cdot z \cdot \mathsf{Adv}^{\mathrm{strong-cpa}}_{\mathcal{B},\mathsf{TTT}_{\mathsf{Limited}}}(\lambda),$$

where $z = z(\lambda)$ is a parameter of the scheme.

Proof. We construct an adversary $\mathcal{B}$ for the strong CPA game for $\mathsf{TTT}_{\mathsf{Limited}}$ using $\mathcal{A}$ as follows:

$\mathsf{Trace}^{D(\cdot)}(\mathsf{pk}, \mathsf{tk}, 1^{1/\epsilon}, \mu_0, \mu_1)$:

1. Construct algorithm $D_1(s, \{c_u\}_{u\in[y]})$ as follows:
   - Sample random $v \leftarrow\$ [z]$. For all $u \in [y]$, set $c_{u,v} := c_u$.
   - For all $v' \in [z] \setminus \{v\}$ and all $u \in [y]$, sample $\mu_{u,v'} \leftarrow\$ \mathcal{M}$ and compute $c_{u,v'} \leftarrow\$ \mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Enc}(\mathsf{pk}, \mu_{u,v'})$.
   - For all $v' \in [z] \setminus \{v\}$, compute $\mu_{v'} := \mu_{1,v'} \oplus \ldots \oplus \mu_{y,v'}$.
   - For all $v' > v$, set $b_{v'} := \langle s, \mu_{v'} \rangle \bmod 2$. For all $v' \leq v$, sample $b_{v'} \leftarrow\$ \{0,1\}$. Define $k := (b_1, \ldots, b_z)$.
   - Sample $\beta \leftarrow\$ \{0,1\}$, and output $\beta \oplus b_v \oplus D(\mu_\beta \oplus k, s, \{c_{u,v}\})$.
2. Construct $D_2(\ \{c_u\}_{u\in[y]}\ ) := \mathsf{GL}^{D_1(\cdot,(c_u)_{u\in[y]})}(\ell, \epsilon' = \epsilon/z)$.
3. Sample $\hat{\mu}_0, \hat{\mu}_1 \leftarrow\$ \mathcal{M}$ and construct $D_3(c)$ as follows:
   - For $w \in 1, \ldots, \zeta := (2yz^3/\Gamma\epsilon^3) \times \omega(\log\lambda)$:
     - Choose a random $u \in [y]$, and set $c_u = c$.
     - Then for each $u' \in [y] \setminus \{u\}$, sample $\mu_{u'} \leftarrow\$ \mathcal{M}$ and run $c_{u'} \leftarrow\$ \mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Enc}(\mathsf{pk}, \mu_{u'})$.
     - Run $\mu' \leftarrow D_2(\ \{c_u\}_{u\in[y]}\ )$, and set $\mu^{(z)} := \mu' \oplus \mu_1 \ldots \oplus \mu_{u-1} \oplus \mu_{u+1} \ldots \oplus \mu_y$.
   - If $\mu^{(w)} = \hat{\mu}_0$ for any $w \in [\zeta]$, output 0. Otherwise, output 1.
4. Return $\mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Trace}^{D_3(\cdot)}(\mathsf{tk}, \mathsf{pk}, \hat{\mu}_0, \hat{\mu}_1)$.

Fig. 17. The Trace algorithm for our generic compiler to get a Threshold traitor tracing scheme TTT from a limited threshold traitor tracing scheme $\mathsf{TTT}_{\mathsf{Limited}}$.

$\mathcal{B}$ receives n, t from $\mathcal{A}$, and it forwards them to its challenger. $\mathcal{B}$ gets back the public keys $\mathsf{pk}, \mathsf{pkc}, \mathsf{tk}$ from its challenger, which it forwards to $\mathcal{A}$. Next, we discuss how $\mathcal{B}$ responds to oracle queries by $\mathcal{A}$:

- $\mathsf{sk}(i)$. $\mathcal{B}$ simply queries its challenger for the key of party i, and forwards the response to $\mathcal{A}$.
- $\mathsf{dec}(\mu, r, i)$. $\mathcal{B}$ parses the randomness r as $(\{\mu_{u,v}, r_{u,v}\}_{u\in[y],v\in[z]}, s)$. For each $u \in [y], v \in [z]$, $\mathcal{B}$ queries its challenger for the oracle $\mathsf{dec}(\mu_{u,v}, r_{u,v}, i)$; let us denote the response by $d'_{u,v,i}$. $\mathcal{B}$ responds to $\mathcal{A}$ with $\{d'_{u,v,i}\}_{u\in[y],v\in[z]}$.

Eventually, $\mathcal{A}$ sends challenge messages μ_0, μ_1 to $\mathcal{B}$. $\mathcal{B}$ samples random messages $\hat{\mu}_0, \hat{\mu}_1 \leftarrow\$ \mathcal{M}$ and sends them to its challenger. Let us use c^* to denote the challenge ciphertext sent to $\mathcal{B}$. $\mathcal{B}$ then generates a challenge ciphertext for $\mathcal{A}$ as follows: It samples a random $v^* \leftarrow\$ [z]$, and random $u^* \leftarrow\$ [y]$, and sets $c_{u^*,v^*} := c^*$. For all $(u, v) \neq (u^*, v^*)$, $\mathcal{B}$ samples $\mu_{u,v} \leftarrow\$ \mathcal{M}$ and $c_{u,v} \leftarrow\$ \mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Enc}(\mathsf{pk}, \mu_{u,v})$. For all $v \neq v^*$, it computes $\mu_v := \mu_{1,v} \oplus \ldots \oplus \mu_{y,v}$. Note that we can define $\mu_{v^*} := \mu_{1,v^*} \oplus \ldots \oplus \mu_{y,v^*}$ the same way, but $\mathcal{B}$ cannot compute it. It then samples $s \leftarrow\$ \mathcal{M}$, and $k^1 \leftarrow\$ \{0,1\}^z$. If $\langle s, \hat{\mu}_0 \rangle = \langle s, \hat{\mu}_1 \rangle \bmod 2$, then $\mathcal{B}$ simply aborts. Otherwise, it computes $b_v := \langle s, \mu_v \rangle \bmod 2$ for all $v \neq v^*$ (b_{v^*} can be defined similarly, but this value is unknown to $\mathcal{B}$). We define a key $k(v)$ as:

$$k(v)_i = \begin{cases} k^1_i & i \leq v \\ b_i & i > v \end{cases}$$

Note that since $\mathcal{B}$ does not know b_{v^*}, it can only compute $k(v)$ for $v \geq v^*$. $\mathcal{B}$ samples $\beta \leftarrow\$ \{0,1\}$, and sets $c := (k(v^*) \oplus \mu_\beta, s, \{c_{u,v}\}_{u\in[y],v\in[z]})$. It sends c to $\mathcal{A}$. Let $\hat{\beta}$ denote $\mathcal{A}$'s response. $\mathcal{B}$ computes $\hat{b} := \beta \oplus k^1_{v^*} \oplus \hat{\beta}$. Lastly, if $\hat{b} = \langle s, \mu_{1,v^*} \oplus \mu_{u^*-1,v^*} \oplus \hat{\mu}_0 \oplus \ldots \oplus \mu_{y,v^*} \rangle \bmod 2$, then $\mathcal{B}$ outputs 0, otherwise it outputs 1.

To analyze the advantage of $\mathcal{B}$, we first define a series of hybrids H_v for $v \in \{0, \ldots, z\}$ as follows. Hybrid H_0 is the original strong CPA game. In hybrid H_v, the challenger runs Setup, and responds to oracle queries exactly as in the original game, but we change how the challenger prepares the

challenge ciphertext. Specifically, it uses $k(v)$ to encrypt the message, more formally, $\hat{c} = k(v) \oplus \mu_\beta$. Here $k(v)$ is as defined above. Let p_v denote the probability with which $\mathcal{A}$ outputs β, i.e. the encrypted message in hybrid H_v. We know that, $p_0 = 1/2 + \mathsf{Adv}^{\mathrm{str-cpa}}_{\mathcal{A},\mathsf{TTT}}(\lambda)$. Moreover, $p_z = 1/2$. This is because, in hybrid H_z, the ciphertext contains $k^1 \oplus \mu_\beta$, which is distributed uniformly random because k^1 is sampled uniformly randomly; meaning that the ciphertext contains no information about β. Hence, in hybrid H_z, the best $\mathcal{A}$ can do is to guess β. Now, looking back at how $\mathcal{B}$ simulates the game to $\mathcal{A}$, we see that it perfectly simulates the public parameters as well as all the oracle queries. Moreover, if $\mathcal{B}$ does not abort, then, we claim that $\mathcal{B}$ essentially simulates either H_{v^*-1} or H_{v^*} to $\mathcal{A}$. To understand why, let us use b to denote the bit sampled by $\mathcal{B}$'s challenger as b. In other words, c^* is an encryption of $\hat{\mu}_b$. We now consider two cases:

- $k^1_{v^*} = b_{v^*}$: More formally, this means that $\mathcal{B}$ was able to correctly guess the v^*th position of the 'real' key b_{v^*}, which is equal to $\langle s, \mu_{v^*} \rangle \bmod 2$. In this case, $\mathcal{B}$ perfectly simulates the hybrid H_{v^*-1}. This is because, the first v^*-1 bits of $k(v^*)$ (as defined above) are distributed randomly, but for all $v \geq v^*$, $k(v^*)_v = b_v$. Hence, in this case, $\mathcal{A}$ will output $\hat{\beta} = \beta$ with probability p_{v^*-1}. This means that, with the same probability, the bit $\hat{b}$ computed by $\mathcal{B}$ will be equal to $k^1_{v^*}$, which by our assumption, is exactly equal to b_{v^*}. i.e. $\hat{b} = \langle s, (\oplus_{u \in [y] \setminus \{u^*\}} \mu_{u,v^*}) \oplus \hat{\mu}_b \rangle \bmod 2$. Hence, $\mathcal{B}$ will correctly guess b in this case with probability p_{v^*-1}.
- $k^1_{v^*} \neq b_{v^*}$: In this case, $\mathcal{B}$ perfectly simulates the hybrid H_{v^*}, since even the v^*th bit of $k(v^*)$ is distributed uniformly at random, independent from the ciphertext. By a similar argument as above, $\hat{b} = k^1_{v^*}$ with probability p_{v^*}. In other words, $\hat{b} = b_{v^*}$ only with probability $1 - p_{v^*}$. Hence, $\mathcal{B}$ will guess correctly with this probability.

in summary, if there is a gap between $p_{v^*} - 1$ and p_{v^*}, then $\mathcal{B}$ can use $\mathcal{A}$ to essentially guess the bit b_{v^*}, which allows it to then guess the bit b. We formalize this intuition in the analysis below: Recall that, $\mu_{v^*} = (\oplus_{u \in [y] \setminus \{u^*\}} \mu_{u,v^*}) \oplus \hat{\mu}_b$. Let E denote the event that $\langle s, \hat{\mu}_0 \rangle = \langle s, \hat{\mu}_1 \rangle \bmod 2$. We now analyze $\mathcal{B}$'s advantage (we use $\mathcal{B}$ outputs b as a shorthand to denote the event that $\mathcal{B}$ correctly guesses the bit b in its strong CPA game):

$$
\begin{aligned}
\mathsf{Adv}^{\mathrm{str-cpa}}_{\mathcal{B},\mathsf{TTT}_{\mathsf{Limited}}}(\lambda) &= \Pr[\mathcal{B} \text{ outputs } b] - \frac{1}{2} \\
&= \frac{1}{2}\left(\Pr[\mathcal{B} \text{ outputs } 0 \mid b = 0] - \Pr[\mathcal{B} \text{ outputs } 0 \mid b = 1]\right) \\
&= \Sigma_{v \in [z]} \frac{1}{2z} \cdot \left(\Pr[\mathcal{B} \text{ outputs } 0 \mid b = 0, v^* = v] - \Pr[\mathcal{B} \text{ outputs } 0 \mid b = 1, v^* = v]\right) \\
&= \Sigma_{v \in [z]} \frac{1}{4z} \cdot \left(\Pr[\mathcal{B} \text{ outputs } 0 \mid b = 0, v^* = v, \neg\mathsf{E}] - \Pr[\mathcal{B} \text{ outputs } 0 \mid b = 1, v^* = v, \neg\mathsf{E}]\right) \\
&= \Sigma_{v \in [z]} \frac{1}{4z} \cdot \begin{pmatrix} \Pr[\mathcal{B} \text{ outputs } b \mid v^* = v, \neg\mathsf{E}, (k^1_{v^*} = b_{v^*})] - \frac{1}{2} + \\ \Pr[\mathcal{B} \text{ outputs } b \mid v^* = v, \neg\mathsf{E}, (k^1_{v^*} \neq b_{v^*})] - \frac{1}{2} \end{pmatrix} \\
&= \Sigma_{v \in [z]} \frac{1}{4z} \cdot (p_{v-1} - 1 + 1 - p_v) \\
&= \frac{p_0 - p_z}{4z} \\
&= \frac{\mathsf{Adv}^{\mathrm{str-cpa}}_{\mathcal{A},\mathsf{TTT}}(\lambda)}{4z}.
\end{aligned}
$$

This proves the theorem.

Traceability. Theorem 23 below proves that the resulting scheme is indeed a secure threshold traitor tracing scheme.

Theorem 23. *For* $y = \omega(\log\lambda), \epsilon' = \epsilon/z, \zeta = (2yz^3/\Gamma\epsilon^3) \times \omega(\log\lambda)$. *Suppose* $\ell = \omega(\lambda)$. *If* $\mathsf{TTT}_{\mathsf{Limited}}$ *is a secure Limited threshold traitor tracing scheme, then* TTT *is a secure (not-Limited) threshold traitor tracing scheme.*

Proof. Fix an adversary $\mathcal{A}$ for TTT and inverse-polynomial ϵ. Let $\mathsf{GoodTr}_{\mathcal{A}}, \mathsf{BadTr}_{\mathcal{A}}, \mathsf{GoodDec}_{\mathcal{A}}$ be the events as in Definition 3 for the compiled scheme TTT. Similarly, let $\mathsf{GoodTr}_{\mathcal{B},\mathsf{lim}}, \mathsf{BadTr}_{\mathcal{B},\mathsf{lim}}$, $\mathsf{GoodDec}_{\mathcal{B},\mathsf{lim}}$ denote these events for the scheme $\mathsf{TTT}_{\mathsf{Limited}}$ for an adversary $\mathcal{B}$.

We start with proving that the event $\mathsf{BadTr}_{\mathcal{A}}$ happens with negligible probability given that $\mathsf{TTT}_{\mathsf{Limited}}$ is a secure Limited threshold traitor tracing scheme. We prove this in Lemma 10.

Lemma 10. *For every* PPT *adversary* $\mathcal{A}$ *for* TTT*, there exists a* PPT *adversary* $\mathcal{B}_1$ *such that,*

$$\Pr[\mathsf{BadTr}_{\mathcal{A}}] \leq \Pr[\mathsf{BadTr}_{\mathcal{B}_1,\mathsf{lim}}]$$

Proof. We construct $\mathcal{B}_1$ which acts as the challenger to $\mathcal{A}$. It receives n, t as input from $\mathcal{A}$, and forwards them to its challenger. $\mathcal{B}_1$ then gets public keys $\mathsf{pk}, \mathsf{pkc}, \mathsf{tk}$ from its challenger, which it forwards to $\mathcal{A}$. On receiving an oracle query $\mathsf{sk}(i)$, $\mathcal{B}_1$ simply queries its challenger for the key of party i, and forwards the response to $\mathcal{A}$.

$\mathcal{A}$ eventually responds with a decoder D^* along with messages μ_0 and μ_1. $\mathcal{B}_1$ then constructs a decoder D_3^* using D^* exactly as in the Trace algorithm specified in Fig. 17. $\mathcal{B}_1$ then sends the messages $\hat{\mu}_0, \hat{\mu}_1$ (sampled when constructing D_3^*) along with D_3^* to its challenger.

It is easy to see that $\mathcal{B}_1$ perfectly simulates the tracing game to $\mathcal{A}$. Moreover, the set of parties corrupted by $\mathcal{B}_1$ is the same as that corrupted by $\mathcal{A}$. Hence, if the event $\mathsf{BadTr}_{\mathcal{A}}$ occurs, meaning that the $\mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Trace}$ algorithm when run on the decoder D_3^* outputs an honest party, the event $\mathsf{BadTr}_{\mathcal{B}_1,\mathsf{lim}}$ also occurs since $\mathcal{B}_1$'s challenger also runs the same Trace algorithm on D_3^*. This proves the lemma.

To show that $\Pr[\mathsf{GoodTr}_{\mathcal{A}}] \geq \Pr[\mathsf{GoodDec}_{\mathcal{A}}] - \mathsf{negl}(\lambda)$, we follow the proof of Theorem 9 in [67]. In more detail, we assume $\mathsf{GoodDec}_{\mathcal{A}}$ happens (D^* guesses b with probability $\geq 1/2+\epsilon$) and analyze the decoders D_1^*, D_2^*, D_3^* constructed by $\mathsf{TTT}.\mathsf{Trace}$. First, we have the following:

Lemma 11. *If* $\mathsf{GoodDec}_{\mathcal{A}}$ *happens, then* $\Pr[D_1^*(s, (c_u)_{u\in[y]}) = \langle s, \mu\rangle \bmod 2] \geq 1/2 + 2\epsilon/z$*, where* $s \leftarrow\!\$\, \mathcal{M} = \{0,1\}^\ell$*,* $\mu_u \leftarrow\!\$\, \mathcal{M}$*,* $c_u \leftarrow\!\$\, \mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Enc}(\mathsf{pk}, \mu_u)$ *for* $u \in [y]$*, and* $\mu = \mu_1 \oplus \cdots \oplus \mu_y$*.*

Proof. If $\mathsf{GoodDec}_{\mathcal{A}}$ happens, then $\Pr[D^*(c^\beta) = \beta] \geq 1/2 + \epsilon$, where $c^\beta \leftarrow\!\$\, \mathsf{TTT}.\mathsf{Enc}(\mathsf{pk}, \mu_\beta)$. Let k denote the key sampled during this call to Enc, i.e. $c^\beta = (\hat{c}^\beta, s, \{c_{u,v}\}_{u\in[y],v\in[z]})$, where $\hat{c}^\beta = \mu_b \oplus k$. Let us sample $k^1 \leftarrow\!\$\, \{0,1\}^z$. Next, for any $v \in [z]$, let $k(v)$ be the following string:

$$k(v)_i = \begin{cases} k_i^1 & i \leq v \\ k_i & i > v \end{cases}$$

Let $c^\beta(v)$ denote the ciphertext encrypting μ_β using $k(v)$. More formally, $c^\beta(v) = (k(v) \oplus \mu_b, s, \{c_{u,v}\}_{u\in[y],v\in[z]})$. Then a routine calculation shows that $\Pr[D^*(c^\beta(v-b)) = b \oplus \beta] \geq 1/2 + \epsilon/z$, where the probability is over a random $v \in [n]$ and random $b \in \{0,1\}$. Notice that the only difference between $D^*(c^\beta(v-1))$ and $D^*(c^\beta(v))$ is that, in the first case the vth bit of the key

is random, whereas in the second case it is k_v. Thus, D^* is in some sense distinguishing k_v from random, with advantage ϵ/z. Then we use the standard trick of turning a bit distinguisher into a bit predictor, by running the distinguisher on a random bit, and then XORing the output with the same random bit. The result is exactly equivalent to running D_1^*, and a routine calculation shows that $\Pr[D_1^*(s,(c_u)_{u\in[y]}) = \langle s,\mu\rangle \bmod 2] \geq 1/2 + 2\epsilon/z$ (where c_u encrypts μ_u and $\mu = \mu_1 \oplus \ldots \oplus \mu_y$) as desired.

Next, the following claim shows that D_2^* actually guesses k:

Lemma 12. *Assuming* $\mathsf{GoodDec}_{\mathcal{A}}$ *happens, then* $\Pr[D_2^*((c_u)_{u\in[y]}) = \mu] \geq \Gamma \times (\epsilon')^3$, *where* $\mu_u \leftarrow\!\$\ \mathcal{M}$, $c_u \leftarrow\!\$\ \mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Enc}(\mathsf{pk},\mu_u)$ *and* $\mu = \mu_1 \oplus \cdots \oplus \mu_y$.

Proof. For this claim, we will use Goldreich-Levin (Theorem 21). We call a ciphertext vector $(c_u)_{u\in[n]}$ "good" if $\Pr[D_1^*(s,(c_u)_{u\in[n]}) = \langle s,\mu\rangle \bmod 2] \geq 1/2 + \epsilon'$, where the probability is over the random choice of s and the random coins in D_1^*. Applying Goldreich-Levin, we have that for any good $(c_u)_{u\in[n]}$, D_2^* will compute k with probability $\Gamma \times (\epsilon')^2$. Lemma 11 states that the overall probability $\Pr[D_1^*(s,(c_u)_{u\in[n]}) = \langle s,\mu\rangle \bmod 2]$ including the randomness of sampling μ_u and c_u is at least $1/2 + 2\epsilon'$. Therefore, with probability at least ϵ', we have that $(c_u)_{u\in[n]}$ is good. Thus, the overall probability D_2^* will compute μ is at least $\Gamma \times (\epsilon')^3$, as desired.

Next, we need to show that D_3^* can decrypt with high probability. Let $\gamma > 0$. Let x denote the random coins for sampling a message in $\mathcal{M}$ along with the secret random coins of $\mathsf{TTT}_{\mathsf{Limited}}$. Define S_γ to be the set of x for $\mathsf{TTT}_{\mathsf{Limited}}$ such that the following experiment, named $\mathsf{CorrectDecrypt}(x)$, outputs 1 with probability at least δ:

- Choose a random $u \in [y]$ and sample μ_u using random coins in x, and compute $c_u \leftarrow \mathsf{TTT}_{\mathsf{Limited}}(\mathsf{pk}, \mu_u; x)$.
- Choose random $\mu_{u'} \leftarrow\!\$\ \mathcal{M}$ and $(c_{u'}) \leftarrow\!\$\ \mathsf{TTT}_{\mathsf{Limited}}(\mathsf{pk},\mu_{u'})$ for each $u' \neq u$.
- Run $\mu \leftarrow\!\$\ D_2^*((c_{u'})_{u'\in[y]})$. Output 1 if $\mu_1 \oplus \mu_2 \oplus \cdots \oplus \mu_y = \mu$.

In other words, S_γ is the set of random coins (which correspond to a message/key pair) such that, if that message/key pair were extended into an entire vector $(c_u)_{u\in[n]}$ by randomly filling in the other ciphertexts and messages, then D_2^* will decrypt correctly i.e. output the XOR of all μ_u with probability at least γ. Let $x_{u'}$ be the random coins used to produce the various $c_{u'}$.

Let η be the fraction of $s \in S_\gamma$. Let p be the overall probability that D_2^* outputs the correct key k. By Claim 5 in [67], we know that, $p \leq \eta^y + y(1-\eta)\gamma$. Using Lemma 12, this means that if $\mathsf{GoodDec}_{\mathcal{A}}$ happens, then $\Gamma(\epsilon')^3 \leq \eta^y + y(1-\eta)\gamma$. We now choose $\gamma = \Gamma(\epsilon')^3/2y$. This implies that $\eta^y \geq \Gamma(\epsilon')^3/2$, meaning $\eta \geq (\Gamma(\epsilon')^3)^{1/y} = (\Gamma\epsilon^3/2y^3)^{1/y}$. By our choice of $y = \omega(\log\lambda)$, since ϵ is inverse polynomial in λ, we have that $\eta \geq \mathsf{poly}(\lambda)(\lambda)^{-1/\omega(\log\lambda)} = 2^{-1/\omega(1)} = 1 - o(1)$.

This means that, conditioned on $\mathsf{GoodDec}_{\mathcal{A}}$, when running D_3^* on a fresh ciphertext encrypting $\hat{\mu}_0$ (which is uniformly sampled from $\mathcal{M}$) from $\mathsf{TTT}_{\mathsf{Limited}}$, the message $\hat{\mu}_0$ and corresponding input ciphertext will come from a "good" set of random coins with probability $\eta = 1 - o(1)$. In this case, every time D_3^* runs D_2^*, the probability D_2^* outputs the correct μ is at least γ. Since $\zeta = \omega(\log\lambda)/\gamma$, the probability D_2^* will output the correct μ in at least one of the runs is then $1-(1-\gamma)^{\omega(\log\lambda)/\gamma} \geq 1 - e^{-\omega(\log\lambda)} = 1 - \mathsf{negl}(\lambda)$, provided the input to D_3^* was good. If D_3^* is given the encryption of $\hat{\mu}_0$ as input, and the ciphertext and message $\hat{\mu}_0$ pair is good, it will correctly output 0 with probability $1 - \mathsf{negl}(\lambda)$. Over all ciphertexts and messages, it will therefore output 0 with probability at least $\gamma(1 - \mathsf{negl}(\lambda)) = 1 - o(1)$ when given the encryption of $\hat{\mu}_0$.

On the other hand, if D_3^* is given the encryption of $\hat{\mu}_1$ as input, then the probability it outputs 0 is the probability the randomly sampled message $\hat{\mu}_0$ matches any one of the outputs of D_2^*. Regardless of whether the input to D_2^* is good or bad, this probability is at most $\zeta/2^\ell$, which is negligible since $\ell = \omega(\lambda)$. Thus, in this case it correctly outputs 1 with probability $1 - \mathsf{negl}(\lambda)$.

The result is that, provided $\mathsf{GoodDec}_{\mathcal{A}}$ happens, D_3^* predicts the correct bit with probability at least $1/2(1 - o(1)) + 1/2(1 - \mathsf{negl}(\lambda))) = 1 - o(1)$. From here, it is straightforward to construct an adversary $\mathcal{B}_2$ from $\mathcal{A}$ which simulates $\mathcal{A}$, except that when $\mathcal{A}$ produces D^*, $\mathcal{B}_2$ will construct and output D_3^*. By the analysis above, $\Pr[\mathsf{GoodDec}_{\mathcal{B}_2,\mathsf{lim}}] = \Pr[\mathsf{GoodDec}_{\mathcal{A}}]$. Moreover, since $\mathsf{TTT.Trace}$ simply outputs whatever $\mathsf{TTT}_{\mathsf{Limited}}.\mathsf{Trace}$ outputs, we have that $\Pr[\mathsf{GoodTr}_{\mathcal{B}_2,\mathsf{lim}}] = \Pr[\mathsf{GoodTr}_{\mathcal{A}}]$. Finally, by the security of $\mathsf{TTT}_{\mathsf{Limited}}$, we have that $\Pr[\mathsf{GoodTr}_{\mathcal{B}_2,\mathsf{lim}}] \geq \Pr[\mathsf{GoodDec}_{\mathcal{B}_2,\mathsf{lim}}] - \mathsf{negl}(\lambda)$, and hence $\Pr[\mathsf{GoodTr}_{\mathcal{A}}] \geq \Pr[\mathsf{GoodDec}_{\mathcal{A}}] - \mathsf{negl}(\lambda)$. This completes the proof.

Acknowledgments. This work was funded by DARPA and NTT Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. We thank David Wu and Dan Boneh for helpful discussions on the paper.

References

1. M. Abdalla, D. Catalano, R. Gay, and B. Ursu. Inner-product functional encryption with fine-grained access control. In S. Moriai and H. Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 467–497, Daejeon, South Korea, Dec. 7–11, 2020. Springer, Cham, Switzerland.
2. S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.
3. M. Ajtai. Generating hard instances of the short basis problem. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP 99*, volume 1644 of *LNCS*, pages 1–9, Prague, Czech Republic, July 11–15, 1999. Springer Berlin Heidelberg, Germany.
4. J. A. Akinyele, C. U. Lehmann, M. D. Green, M. W. Pagano, Z. N. J. Peterson, and A. D. Rubin. Self-protecting electronic medical records using attribute-based encryption. Cryptology ePrint Archive, Report 2010/565, 2010.
5. J. Alwen and C. Peikert. Generating shorter bases for hard random lattices. Cryptology ePrint Archive, Report 2008/521, 2008.
6. B. Applebaum, O. Nir, and B. Pinkas. How to recover a secret with $o(n)$ additions. In H. Handschuh and A. Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 236–262, Santa Barbara, CA, USA, Aug. 20–24, 2023. Springer, Cham, Switzerland.
7. K. Asano, N. Attrapadung, K. Hara, K. Hashimoto, and Y. Watanabe. Key revocation in registered attribute-based encryption. In T. Jager and J. Pan, editors, *PKC 2025, Part III*, volume 15676 of *LNCS*, pages 102–133, Røros, Norway, May 12–15, 2025. Springer, Cham, Switzerland.
8. M. Ball, A. Çakan, and T. Malkin. Linear threshold secret-sharing with binary reconstruction. In S. Tessaro, editor, *ITC 2021*, volume 199 of *LIPIcs*, pages 12:1–12:22, Seattle, WA, USA, July 23–26, 2021. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.
9. J. Bebel and D. Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Report 2022/898, 2022.
10. R. Bhattacharyya, J. Bormet, S. Faust, P. Mukherjee, and H. Othman. CCA-secure traceable threshold (ID-based) encryption and application. Cryptology ePrint Archive, Paper 2025/341, 2025.
11. R. Bhattacharyya, J. Bormet, S. Faust, P. Mukherjee, and H. Othman. CCA-secure traceable threshold (ID-based) encryption and application. Cryptology ePrint Archive, Report 2025/341, 2025.
12. D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Cham, Switzerland.
13. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In P. Q.

Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.
14. D. Boneh, A. Partap, and L. Rotem. Accountability for misbehavior in threshold decryption via threshold traitor tracing. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 317–351, Santa Barbara, CA, USA, Aug. 18–22, 2024. Springer, Cham, Switzerland.
15. D. Boneh, A. Partap, and L. Rotem. Traceable secret sharing: Strong security and efficient constructions. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part V*, volume 14924 of *LNCS*, pages 221–256, Santa Barbara, CA, USA, Aug. 18–22, 2024. Springer, Cham, Switzerland.
16. D. Boneh, A. Partap, and L. Rotem. Traceable verifiable random functions. Cryptology ePrint Archive, Report 2025/312, 2025.
17. D. Boneh, A. Sahai, and B. Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 573–592, St. Petersburg, Russia, May 28 – June 1, 2006. Springer Berlin Heidelberg, Germany.
18. J. Bormet, A. R. Choudhuri, S. Faust, S. Garg, H. Othman, G.-V. Policharla, Z. Qu, and M. Wang. BEAST-MEV: Batched threshold encryption with silent setup for MEV prevention. Cryptology ePrint Archive, Paper 2025/1419, 2025.
19. J. Bormet, J. Hofmann, and H. Othman. Traceable threshold encryption without trusted dealer. Cryptology ePrint Archive, Report 2025/342, 2025.
20. K. Boudgoust and P. Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In J. Guo and R. Steinfeld, editors, *ASIACRYPT 2023, Part I*, volume 14438 of *LNCS*, pages 371–404, Guangzhou, China, Dec. 4–8, 2023. Springer, Singapore, Singapore.
21. Z. Brakerski and S. Medina. Limits on adaptive security for attribute-based encryption. In E. Boyle and M. Mahmoody, editors, *TCC 2024, Part III*, volume 15366 of *LNCS*, pages 91–123, Milan, Italy, Dec. 2–6, 2024. Springer, Cham, Switzerland.
22. S. Canard, N. Papon, and D. H. Phan. Public traceability in threshold decryption. Cryptology ePrint Archive, Paper 2025/1347, 2025.
23. G. Castagnos, F. Laguillaumie, and I. Tucker. Threshold linearly homomorphic encryption on $\mathbf{Z}/2^k\mathbf{Z}$. In S. Agrawal and D. Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 99–129, Taipei, Taiwan, Dec. 5–9, 2022. Springer, Cham, Switzerland.
24. R. Chairattana-Apirom, S. Tessaro, and C. Zhu. Partially non-interactive two-round lattice-based threshold signatures. In K.-M. Chung and Y. Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 268–302, Kolkata, India, Dec. 9–13, 2024. Springer, Singapore, Singapore.
25. A. Chakrabarti, M. Maitra, A. Mondal, and K. Sehgal. Silent threshold traitor tracing & enhancing mempool privacy. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, page 1769–1783, New York, NY, USA, 2025. Association for Computing Machinery.
26. M. Chase and S. S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM CCS 2009*, pages 121–130, Chicago, Illinois, USA, Nov. 9–13, 2009. ACM Press.
27. J. H. Cheon, W. Cho, and J. Kim. Improved universal thresholdizer from iterative shamir secret sharing. Cryptology ePrint Archive, Report 2023/545, 2023.
28. A. R. Choudhuri, S. Garg, J. Piet, and G.-V. Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In D. Balzarotti and W. Xu, editors, *USENIX Security 2024*, Philadelphia, PA, USA, Aug. 14–16, 2024. USENIX Association.
29. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 103–118, Konstanz, Germany, May 11–15, 1997. Springer Berlin Heidelberg, Germany.
30. M. Dahl, D. Demmler, S. El Kazdadi, A. Meyre, J.-B. Orfila, D. Rotaru, N. P. Smart, S. Tap, and M. Walter. Noah's ark: Efficient threshold-FHE using noise flooding. Cryptology ePrint Archive, Report 2023/815, 2023.
31. S. Das, L. Ren, and Z. Yang. Adaptively secure threshold ElGamal decryption from DDH. Cryptology ePrint Archive, Paper 2025/1477, 2025.
32. P. Datta, I. Komargodski, and B. Waters. Decentralized multi-authority ABE for DNFs from LWE. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 177–209, Zagreb, Croatia, Oct. 17–21, 2021. Springer, Cham, Switzerland.
33. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533, Montréal, Québec, Canada, May 23–25, 1994. ACM Press.
34. H. de Valence. The Penumbra protocol. link.

35. Y. Desmedt. Society and group oriented cryptography: A new concept. In C. Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127, Santa Barbara, CA, USA, Aug. 16–20, 1988. Springer Berlin Heidelberg, Germany.
36. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315, Santa Barbara, CA, USA, Aug. 20–24, 1990. Springer, New York, USA.
37. J. Devevey, B. Libert, K. Nguyen, T. Peters, and M. Yung. Non-interactive CCA2-secure threshold cryptosystems: Achieving adaptive security in the standard model without pairings. In J. Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 659–690, Virtual Event, May 10–13, 2021. Springer, Cham, Switzerland.
38. X. T. Do, D. H. Phan, and D. Pointcheval. Traceable inner product functional encryption. In S. Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 564–585, San Francisco, CA, USA, Feb. 24–28, 2020. Springer, Cham, Switzerland.
39. Y. Frankel. A practical protocol for large group oriented networks. In J.-J. Quisquater and J. Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 56–61, Houthalen, Belgium, Apr. 10–13, 1990. Springer Berlin Heidelberg, Germany.
40. M. Gagné, S. Narayan, and R. Safavi-Naini. Threshold attribute-based signcryption. In J. A. Garay and R. De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 154–171, Amalfi, Italy, Sept. 13–15, 2010. Springer Berlin Heidelberg, Germany.
41. S. Garg, D. Kolonelos, G.-V. Policharla, and M. Wang. Threshold encryption with silent setup. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 352–386, Santa Barbara, CA, USA, Aug. 18–22, 2024. Springer, Cham, Switzerland.
42. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 197–206, Victoria, BC, Canada, May 17–20, 2008. ACM Press.
43. O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32, Seattle, WA, USA, May 15–17, 1989. ACM Press.
44. J. Gong, J. Luo, and H. Wee. Traitor tracing with $N^{1/3}$-size ciphertexts and $O(1)$-size keys from k-Lin. In C. Hazay and M. Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 637–668, Lyon, France, Apr. 23–27, 2023. Springer, Cham, Switzerland.
45. R. Goyal, V. Koppula, A. Russell, and B. Waters. Risky traitor tracing and new differential privacy negative results. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 467–497, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Cham, Switzerland.
46. R. Goyal, V. Koppula, and B. Waters. Collusion resistant traitor tracing from learning with errors. In I. Diakonikolas, D. Kempe, and M. Henzinger, editors, *50th ACM STOC*, pages 660–670, Los Angeles, CA, USA, June 25–29, 2018. ACM Press.
47. V. Goyal, Y. Song, and A. Srinivasan. Traceable secret sharing and applications. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 718–747, Virtual Event, Aug. 16–20, 2021. Springer, Cham, Switzerland.
48. K. D. Gür, J. Katz, and T. Silde. Two-round threshold lattice-based signatures from threshold homomorphic encryption. In M.-J. Saarinen and D. Smith-Tone, editors, *Post-Quantum Cryptography - 15th International Workshop, PQCrypto 2024, Part II*, pages 266–300, Oxford, UK, June 12–14, 2024. Springer, Cham, Switzerland.
49. M. Hall-Andersen, M. Simkin, and B. Wagner. Silent threshold encryption with one-shot adaptive security. Cryptology ePrint Archive, Paper 2025/1384, 2025.
50. J. Herranz, F. Laguillaumie, and C. Ràfols. Constant size ciphertexts in threshold attribute-based encryption. In P. Q. Nguyen and D. Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 19–34, Paris, France, May 26–28, 2010. Springer Berlin Heidelberg, Germany.
51. S. Hohenberger, G. Lu, B. Waters, and D. J. Wu. Registered attribute-based encryption. In C. Hazay and M. Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 511–542, Lyon, France, Apr. 23–27, 2023. Springer, Cham, Switzerland.
52. Y. Ishai and H. Wee. Partial garbling schemes and their applications. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 650–662, Copenhagen, Denmark, July 8–11, 2014. Springer Berlin Heidelberg, Germany.
53. M. Karchmer and A. Wigderson. On span programs. In *Proceedings of Structures in Complexity Theory*, pages 102–111, 1993.
54. G. Lu, B. Waters, and D. J. Wu. Multi-authority registered attribute-based encryption. In S. Fehr and P.-A. Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 3–33, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

55. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718, Cambridge, UK, Apr. 15–19, 2012. Springer Berlin Heidelberg, Germany.
56. P. Mukherjee. Adaptively secure threshold symmetric-key encryption. In K. Bhargavan, E. Oswald, and M. Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 465–487, Bangalore, India, Dec. 13–16, 2020. Springer, Cham, Switzerland.
57. S. Müller, S. Katzenbeisser, and C. Eckert. Distributed attribute-based encryption. In P. J. Lee and J. H. Cheon, editors, *ICISC 08*, volume 5461 of *LNCS*, pages 20–36, Seoul, Korea, Dec. 3–5, 2009. Springer Berlin Heidelberg, Germany.
58. M. Naor and B. Pinkas. Threshold traitor tracing. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 502–517, Santa Barbara, CA, USA, Aug. 23–27, 1998. Springer Berlin Heidelberg, Germany.
59. Osmosis. link.
60. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow and R. Fagin, editors, *37th ACM STOC*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.
61. K. Sako. An auction protocol which hides bids of losers. In H. Imai and Y. Zheng, editors, *PKC 2000*, volume 1751 of *LNCS*, pages 422–432, Melbourne, Victoria, Australia, Jan. 18–20, 2000. Springer Berlin Heidelberg, Germany.
62. Shutter: Preventing front running and malicious MEV on ethereum. link.
63. S. Suegami, S. Ashizawa, and K. Shibano. Constant-cost batched partial decryption in threshold encryption. Cryptology ePrint Archive, Paper 2024/762, 2024.
64. M. Venema. A practical compiler for attribute-based encryption: New decentralized constructions and more. In M. Rosulek, editor, *CT-RSA 2023*, volume 13871 of *LNCS*, pages 132–159, San Francisco, CA, USA, Apr. 24–27, 2023. Springer, Cham, Switzerland.
65. B. Waters and D. Wichs. Adaptively secure attribute-based encryption from witness encryption. In E. Boyle and M. Mahmoody, editors, *TCC 2024, Part III*, volume 15366 of *LNCS*, pages 65–90, Milan, Italy, Dec. 2–6, 2024. Springer, Cham, Switzerland.
66. B. Waters and D. J. Wu. Silent threshold cryptography from pairings: Expressive policies in the plain model. Cryptology ePrint Archive, Paper 2025/1547, 2025.
67. M. Zhandry. New techniques for traitor tracing: Size $N^{1/3}$ and more from pairings. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 652–682, Santa Barbara, CA, USA, Aug. 17–21, 2020. Springer, Cham, Switzerland.
68. M. Zhandry. Optimal traitor tracing from pairings. In S. Fehr and P.-A. Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 305–335, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.