Threshold Encryption with Silent Setup

Sanjam Garg[1], Dimitris Kolonelos[*2,3], Guru-Vamsi Policharla[1], and Mingyuan Wang[1]

[1]UC Berkeley, {sanjamg,guruvamsip,mingyuan}@berkeley.edu
[2]IMDEA Software Institute, dimitris.kolonelos@imdea.org
[3]Universidad Politécnica de Madrid

Abstract

We build a concretely efficient threshold encryption scheme where the joint public key of a set of parties is computed as a *deterministic* function of their locally computed public keys, enabling a *silent* setup phase. By eliminating interaction from the setup phase, our scheme immediately enjoys several highly desirable features such as asynchronous setup, multiverse support, and dynamic threshold.

Prior to our work, the only known constructions of threshold encryption with silent setup relied on heavy cryptographic machinery such as indistinguishability Obfuscation or witness encryption for all of NP. Our core technical innovation lies in building a special purpose witness encryption scheme for the statement "at least t parties have signed a given message". Our construction relies on pairings and is proved secure in the Generic Group Model.

Notably, our construction, restricted to the special case of threshold $t = 1$, gives an alternative construction of the (flexible) distributed broadcast encryption from pairings, which has been the central focus of several recent works.

We implement and evaluate our scheme to demonstrate its concrete efficiency. Both encryption and partial decryption are constant time, taking < 7 ms and < 1 ms, respectively. For a committee of 1024 parties, the aggregation of partial decryptions takes < 200 ms, when all parties provide partial decryptions. The size of each ciphertext is $\approx 8\times$ larger than an ElGamal ciphertext.

[*]Most of the work was done while the second author was a visiting scholar at UC Berkeley.

Contents

1 Introduction

Threshold encryption [Des88, DF90] is a fundamental cryptographic primitive that allows an encryptor to generate a *succinct* ciphertext such that it can be decrypted by any threshold t sized subset of n parties, while remaining semantically secure against any coalition of up to $t-1$ parties. Typically, a threshold encryption system begins with an *interactive* setup phase where a distributed key generation (DKG) protocol is run to establish a public key pair, where the secret key is shared amongst the n parties. Any party can then encrypt a message using the public key and produce a *succinct* ciphertext, of size independent of the number of parties. It is highly desirable for the decryption process to be *non-interactive*, i.e., the parties locally produce partial decryptions of the ciphertext, which can be *publicly* aggregated to recover the message.

Expensive DKG. Although the original notion was proposed over three decades ago, and there has been a long line of research on this topic, virtually all known threshold encryption schemes[1] require a DKG protocol to sample a correlated partial decryption key for each party. While this is theoretically feasible, DKG protocol in practice is quite expensive. For instance, it typically requires high communication/computation complexity [GJKR99, TCZ+20]. Moreover, these theoretical solutions typically assume a synchronous setting. For the more practically-relevant asynchronous setting, despite many efforts [AJM+21, KKMS20, KMS20, DXR21, DYX+22], these costs remains even higher. Moreover, these asynchronous protocols can only tolerate $< 1/3$ fraction of malicious corruptions. Therefore, for many practical applications (refer to Section 1.2), it is much desirable if the setup phase of threshold encryption schemes can be made entirely non-interactive.

Multiverse Support. One may also want the ability to add or remove a party from the committee without any additional interaction between the parties. This is particularly useful in practice, where one may want to easily onboard new parties or remove unresponsive members from the committee. This feature is absent is traditional threshold encryption schemes and any changes require (at least a threshold number of) committee members to be online. Instead, it would be highly desirable if a one-time setup could enable, *without any additional interaction*, the setup of all future threshold encryption for different universes (i.e., the multiverse setting introduced for signatures in [BGJ+23, GJM+24]).

Dynamic Threshold. Another desirable property is to allow encrypting parties to choose a different threshold for each ciphertext without the need for repeating the interactive setup. This allows for a flexible tradeoff between security and liveness. For instance, a party can choose a higher threshold if they are willing to tolerate a higher risk of decryption failure (say due to offline parties), in exchange for reducing the trust in the committee. Typically, any new threshold would require parties to engage in a new instance of DKG, and the committee member's secret state will grow with the number of thresholds maintained.

Although there have been prior attempts to achieve the latter two properties, they either require a dedicated trusted party (typically called the Private Key Generator) to generate and pass the secret keys of the users [DP08, HLR10] or the size of the ciphertext is linear in committee size [DHMR08, DHMR07]. The only known solution with constant sized ciphertexts uses $i\mathcal{O}$ [RSY21]. In this work, we ask:

Can we realize threshold encryption without an interactive setup phase?

In particular, we want constant-size ciphertexts and non-interactive decryption. Similar questions have been recently asked for threshold signatures [GJM+24, DCX+23] or threshold encryption restricted to

[1] The only exception we are aware of is [RSY21], which relies on Indistinguishability Obfuscation ($i\mathcal{O}$) [BGI+01].

the special case of threshold $t = 1$ (a.k.a., distributed broadcast encryption[2]) [WQZD10, BZ14, FWW23, KMW23, GLWW23]; however, this question has remained unexplored for threshold encryption.

1.1 Our Contributions

Silent Threshold Encryption. As our first contribution, we propose the notion of silent threshold encryption (STE). In STE, all parties locally sample a public key pair $\{(\mathsf{sk}_i, \mathsf{pk}_i)\}_{i\in[n]}$. These public keys can be *publicly* aggregated in a *deterministic* manner to produce a *succinct* encryption key ek. Importantly, ek is the only information required to encrypt a message. The threshold number of parties required to decrypt a ciphertext can be chosen *at the time of encryption*. As a crucial efficiency requirement, the encryption key ek, the ciphertext ct, and partial decryptions σ_i should all be of constant size. The aggregation time for recovering the message from partial decryptions should be both asymptotically (i.e., linear in the number of parties) and concretely comparable to standard threshold encryption.

STE naturally achieves all of the properties discussed above. It enjoys 1) silent setup – no interaction is required at all, 2) multiverse – a one-time setup that enables all future universe generation, and 3) dynamic threshold – every ciphertext comes with a ciphertext-specific threshold.

We emphasize that, before our work, the only known path to build STE used heavy cryptographic machinery such as indistinguishability Obfuscation ($i\mathcal{O}$) [BGI+01] or witness encryption for all NP [GGH+13].[3] In fact, various related primitives have been studied in the literature, such as distributed broadcast encryption [WQZD10, BZ14] and threshold broadcast encryption [RSY21]. Although some of these works studied a weaker variant of our primitive, all of their constructions require strong assumptions ($i\mathcal{O}$ and witness encryption). We refer the readers to Section 1.2 and Section 1.3 for more detailed discussions on this. Our work directly gives the first (concretely efficient) construction of this primitive based on pairing-friendly groups, which we discuss next.

A practical STE scheme. Our construction assumes a common reference string (CRS), which is similar to the CRS used in the KZG polynomial commitment scheme [KZG10]. We prove the security of our scheme in the generic group model (GGM) [Sho97, Mau05].

Stated simply, our approach starts with committee members sampling a signature key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ and having them publish pk_i. When encrypting a message, a random ciphertext-specific tag is sampled, which can be viewed as a random string. We then build a witness encryption scheme[4] for the following statement: "I have valid signatures under t-out-of-n public keys $\{\mathsf{pk}_i\}_{i\in[n]}$ on the (ciphertext-specific) tag". During decryption, committee members simply sign the tag as their partial decryption, using which, the corresponding ciphertext can be decrypted. The sematic security is guaranteed against any collusion of $< t$ number of parties.

In more detail, the signature scheme we use is a modification of the silent threshold signature scheme recently introduced in [GJM+24] (also concurrently [DCX+23]). Notably, our modification actually improves the efficiency of the original scheme [GJM+24]. In particular, the verification of the aggregated signature does not use any Fiat-Shamir heuristics [FS87], which is also crucial for our construction.

Arbitrary Threshold in the Asynchronous Setting. In typical threshold encryption schemes, the secret key is first shared using a linear-secret sharing scheme during the interactive setup phase which

[2] That is, distributed broadcast encryption can be seen as a setup-free threshold encryption supporting multiverse with threshold $t = 1$.

[3] This is in sharp contrast to threshold signatures with silent setup [GJM+24, DCX+23], where, theoretically, one could always apply a succinct non-interactive argument of knowledge (SNARK) to obtain a non-black-box solution.

[4] Throughout our work we slightly abuse the term 'witness encryption(WE)'. As a matter fact, we mostly refer to a relaxed type of WE where semantic security holds even when the witness is only computationally hard to find (in contrast to statistically in the original WE notion [GGH+13]).

requires a DKG. In the asynchronous network setting, this limits the maximum corruption threshold to be $t < n/3$. In contrast, our scheme is the first, practical, threshold encryption scheme that can tolerate arbitrary corruption threshold in the asynchronous setting as we completely avoid the DKG and only need a PKI where parties register their public key.

Implementation and Evaluation. We created a Rust crate containing an implementation of our silent-threshold encryption scheme. Our benchmarks reveal that threshold encryption with silent setup is indeed practical. The ciphertext size is 9 group elements (768 bytes), which is only $8\times$ as large as an ElGamal ciphertext. The encryption time is < 7 ms, which is independent of the committee size. For a maximum committee size of 1024 parties, it takes < 28 s for parties to set up their public keys. This is a one-time cost, and our implementation can be optimized further. Partial decryption takes < 1 ms as it requires just one group operation, irrespective of committee size. Finally, given partial decryptions, the message can be recovered in ≈ 200 ms for a committee size of 1024 parties. In large-scale distributed networks where DKGs can be very expensive, thereby limiting committee sizes to small numbers, we argue that our scheme offers a viable path for scaling to large committee sizes.

1.2 Applications

Advanced Encryption Schemes. Our work also provides new constructions and insights into many other advanced encryption primitives. We highlight them next.

1. *Distributed Broadcast Encryption.* Broadcast encryption [FN94] allows encryption to a subset $S \subseteq [n]$ (of parties). The security requirement is that parties can decrypt the message if and only if they belong to the target universe S. Crucially, the ciphertext should be succinct, ideally independent of the set size $|S|$. Traditionally, broadcast encryption considers the setting, where a central trusted party (the Private Key Generator) distributes secret keys for each party. A distributed broadcast encryption (DBE) [WQZD10, BZ14], on the other hand, asks for the same functionality, while also demanding a silent setup, without any central trusted party. That is, parties locally sample their secret/public key pairs. For more than a decade, the only constructions of distributed broadcast encryption either relied on indistinguishability obfuscation ($i\mathcal{O}$) [BZ14] or came without formal security arguments [WQZD10]. Only very recently, two works showed constructions of this primitive from simpler assumptions [FWW23, KMW23]. Our work provides an alternative solution to this problem. Indeed, distributed broadcast encryption is a special case of threshold encryption with silent setup, where the threshold $t = 1$.[5] In particular, the concrete efficiency of our scheme is comparable to the state-of-the-art [KMW23], which is also based on pairings.

2. *Flexible Broadcast Encryption.* Recently Freitag et al. introduced the notion of Flexible Broadcast Encryption (FBE) [FWW23], which is, in essence, a stronger variant of Distributed Broadcast Encryption (DBE) where users are oblivious of the state of the system at the time of their local keys sampling. In particular, in DBE users keys' are associated with a unique index, typically a counter of the users currently in the system, unlike FBE where keys are statelessly sampled. Freitag et al. showed a construction of FBE from Witness Encryption. Concurrently to our work, Garg et al. [GLWW23] showed a compiler to boost any DBE scheme to an FBE, therefore in combination with [KMW23] achieved an FBE from pairings. However their compiler induces an $\omega(\log \lambda)$ overhead on the size of each public key. We show that a slight

[5] In fact, threshold encryption ($t = 1$) with silent setup is slightly stronger than distributed broadcast encryption in that it decouples the generation of encryption key ek for a universe S and the encryption step. This one-time cost of computing ek could be amortized if one wants to broadcast many messages to the same universe S. In distributed broadcast encryption, this is not necessarily the case.

modification of our STE scheme provides a direct construction of FBE *without any overhead on the size of the public keys*. Therefore, this comprises the first FBE from pairings with $O(1)$ (in group elements) public keys' size. We stress that this is not the focus of our work, but, notably, our techniques allow us to get this highly desired feature for free.

3. *Threshold Broadcast Encryption.* More generally, one may consider a broadcast encryption with threshold $t > 1$. That is, the message can only be decrypted if and only if $\geqslant t$ parties from the target universe S partially decrypt it. This primitive is studied by several prior works under different names (e.g., dynamic threshold encryption [DP08] and threshold broadcast encryption [DHMR07, HLR10, RSY21]). Note that this notion is slightly weaker than our notion of *multiverse threshold encryption*.[6] Unlike broadcast encryption, there is no work that realized threshold broadcast encryption with a silent setup – either the size of the ciphertext is linear in $|S|$ [DHMR08, DHMR07] or they require a trusted setup [DP08, HLR10], with the only exception being [RSY21] that resorts to $i\mathcal{O}$. Our work is the first one to construct a practical, threshold broadcast encryption with a silent setup.

Mempool Transaction Privacy. In many popular blockchains, including Ethereum (the chain with the largest DeFi liquidity), transactions from users are first submitted to a public mempool from which miners select transactions and create blocks that are to be appended to the blockchain. During the process, miners are free to insert their own transactions before and/or after a user's transactions, thereby allowing them to frontrun/backrun other transactions. This provides users of Decentralized Exchanges with worse prices, hurting the users' experience. This phenomenon was first documented under the umbrella of *Miner Extractable Value* (MEV), [DGK+20] with many followup works showing that it is a widespread issue [GKW+16, QZG21, TC+21, JSSW21, CJW22]. In particular, an estimated 200,000,000 USD were lost on Ethereum in 2021 alone, mostly benefiting miners [PFW22].

Consider another situation where a user discovers a bug in their smart contract that allows any party to drain all of its funds. A natural course of action could be to first *recover* the funds before any other party drains the account. However, when this user submits a transaction to the public mempool, a miner can *copy* this transaction to make themselves the receiver, and bribe other miners/pay higher gas fees to be included before the honest user. Finally, revealing the content of transactions allows miners to selectively censor certain users or certain types of transactions.

By encrypting transactions, the above issues can be mitigated as miners cannot frontrun transactions they have no information about. Encrypted mempools [BO22, PNS23, MS22, KLJD23, RK23, DHMW23] have been a topic of active research in recent years, but they all require an expensive setup procedure limiting the committee sizes. Our solution completely avoids the setup procedure and is non-interactive, thereby allowing committees to scale even further.

Finally, in many applications, encryption and signatures are often used together – signatures for authentication and encryption for confidentiality. This is, in particular, true in the threshold setting. In blockchain applications, for instance, signatures are used for validating blocks and encryptions are used for the confidentiality of the transactions. Thus, it would be ideal if the same system supports both encryption and signatures. Otherwise, two independent systems need to be implemented. This was done as part of McFly [DHMW23], where the threshold signature functionality was augmented to support threshold encryption. However, their system needs a DKG setup. The solution we develop in this work achieves both signatures and encryption in one system without setup.

[6] Similar to the case of Distributed Broadcast Encryption, the difference is the decoupling of the encryption key derivation and the encryption step.

1.3 Related Work

Removing DKG from Threshold Crypto. DKG has been a bottleneck for deploying threshold signatures at scale for a long time. For instance, Ethereum 2.0 periodically samples 512 validators to sign on the newly created blocks to reach consensus [eth]. Even for this moderate universe size, DKG is too costly that they opt for a multisignature, mainly due to the advantage of having a non-interactive setup. For the purpose of removing DKG, several recent works [MRV+21, GJM+24, DCX+23] gave various solutions to construct threshold signatures with a silent setup.

Note that, even in the silent setup setting where parties sample their key pairs independently, one can always generically apply existing succinct non-interactive arguments (SNARKs) to construct threshold signature schemes. That is, the aggregator will produce a SNARK proof certifying the statement: $\geqslant t$ parties have signed the message. With the recent rapid development in SNARK literature [Gro16, GWC19], such generic constructions will have small aggregated signatures and extremely fast verification time. The bottleneck, however, lies in proof generation time, which is the signature aggregation time for threshold signature. Therefore, the recent research efforts [MRV+21, GJM+24, DCX+23] can be viewed as designing custom SNARK schemes for signature verification with concretely efficient aggregation time.

Contrary to threshold signatures, removing DKG from threshold encryption is a *significantly more difficult* task. Note that, unlike threshold signature, there are *no generic feasibility solutions*. That is, it is not even clear if there are theoretical solutions regardless of the concrete efficiency. In fact, the only solution for threshold encryption with silent setup in the literature [RSY21] requires $i\mathcal{O}$. As we have already discussed in Section 1.2, even for the restricted setting of $t = 1$ and the closely related notion of distributed broadcast encryption, we only recently began to have feasibility solutions [FWW23, KMW23].

Removing Setup from Advanced Encryption. More broadly, many recent works have been trying to remove the trusted setup in different advanced encryption schemes to move to a silent setup setting. In Section 1.2, we have already discussed the works [WQZD10, BZ14, FWW23, KMW23, GLWW23] that construct broadcast encryption with a silent setup (i.e., distributed broadcast encryption). For identity-based encryption (IBE), the work of [GHMR18] initiated the study of registration-based encryption (RBE) as an IBE with a silent setup. A long line of works has been trying to construct concretely efficient RBE schemes [GHM+19, GV20, CES21, GKMR23, DKL+23, FKdP23]. Moreover, several recent works have extended this research effort to other advanced encryption schemes, such as (registered) attribute-based encryption [HLWW23, ZZGQ23] and (registered) functional encryption [FFM+23, DP23].

Our work belongs to this line of research, which initiates the removal of setup for the case of threshold encryption.

2 Technical Overview

Our objective is to construct a silent threshold encryption (STE), where parties independently sample their key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$. Afterward, a *succinct* encryption key ek can be *deterministically* derived from the public keys $\{\mathsf{pk}_i\}_i$. Given any message msg and a message-specific threshold t, the encryptor can produce a ciphertext $\mathsf{ct} = \mathsf{Enc}(\mathsf{ek}, \mathsf{msg}, t)$. Given any ct, party can locally partial decrypt it as $\sigma_i = \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$. We emphasize that the ciphertext ct and partial decryptions σ_i are also required to be constant-size. Given enough ($\geqslant t$) partial decryptions, the correct message can be reconstructed.

As we have already discussed, although STE is a natural strengthening of "threshold encryption with an interactive setup" (e.g., DKG), this notion turns out to be remarkably hard to construct (even for $t = 1$). On the other hand, silent threshold signature (STS) [GJM+24, DCX+23] is relatively easier to construct. Hence, our work starts with the following question.

Can we leverage a silent threshold signature
to construct a silent threshold encryption?

Signature-based witness encryption (SWE) [DHMW23] is a generic tool for building encryption schemes from signature schemes. It allows the encryptor to encrypt a message with respect to the statement that there is a valid (aggregated) signature under some tag. The security guarantees that one can recover the message if and only if it does hold such a valid signature.

Although this approach conceptually works, many technical challenges remains. We do not know how to construct SWE for an arbitrary signature scheme. [DHMW23] only shows how to construct SWE for threshold BLS signatures [BLS01]. In general, (plain) witness encryption for any NP languages typically requires strong assumptions such as $i\mathcal{O}$ [GGH+13] or non-standard non-falsifiable (knowledge) lattice assumptions [Tsa22, VWW22]. Therefore, to instantiate this conceptual plan, we must overcome the following technical challenges.

1. *Construct an SWE for a* large class *of signatures.*
2. *Construct an STS that falls into this class.*

2.1 Signature-based Witness Encryption for Linear Verifiable Signature

The starting point of our construction is to realize that one can construct an SWE for any signature scheme whose verification is a *public linear constraint* system. Take BLS signature [BLS01] (refer to Definition 1) as an example, the verification checks if

$$g \circ \sigma \stackrel{?}{=} \mathsf{pk} \circ \mathsf{RO}(\mathsf{tag}).^{7}$$

Here, the signature is σ, and everything else is *public*. Crucially, the verification is checking *a linear function* in the signature σ (as in it never computes $\sigma \circ \sigma$). Given such a linear verification, one may witness encrypt a message msg as

$$\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = \Big(\alpha \cdot g,\ \alpha \cdot (\mathsf{pk} \circ \mathsf{RO}(\mathsf{tag})) + \mathsf{msg} \Big),^{8}$$

where α is a random field element sampled by the encryptor. Given the signature σ, the decryption is done by $\mathsf{ct}_2 - (\mathsf{ct}_1 \circ \sigma)$. An astute reader might realize that this is Boneh-Franklin identity-based encryption [BF01] from a different perspective.

More generally, one may imagine a more sophisticated signature scheme with a linear verification (in matrix form) as

$$A_{u \times v} \ \circ \ \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_v \end{pmatrix} \ \stackrel{?}{=} \ \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_u \end{pmatrix}.^{9} \tag{1}$$

In this particular example, the signature consists of v group elements $(\sigma_1, \ldots, \sigma_v)$ and the verification checks u pairing equations. Most crucially, the matrix A and the vector $(b_1, \ldots, b_u)^\intercal$ are public information

[7] Here, RO stands for the random oracle, and $\circ$ stands for the pairing operation. For simplicity, we present it with symmetric pairing for now, but everything works similarly for asymmetric pairing.

[8] We adopt additive notation for the standard group operation.

[9] $A_{u \times v}$ highlights the fact that the dimension of A is $u \times v$.

given the tag to sign. For such a signature scheme, one may witness encrypt it as

$$\begin{aligned}\mathsf{ct} &= \left((\alpha_1, \ldots, \alpha_u) \cdot A,\ (\alpha_1,\ \ldots, \alpha_u) \cdot \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_u \end{pmatrix} + \mathsf{msg} \right) \\ &= \Big((\alpha_1, \ldots, \alpha_u) \cdot A,\ \alpha_1 \cdot b_1 + \cdots + \alpha_u \cdot b_u + \mathsf{msg} \Big).\end{aligned}$$

Again, $\alpha_1, \ldots, \alpha_u$ are random field elements sampled by the encryptor. Here, the ciphertext consists of $v+1$ group elements and, given a valid signature $(\sigma_1, \ldots, \sigma_v)$, one may decrypt is as

$$\mathsf{ct}_{v+1} - \left((\mathsf{ct}_1, \ldots, \mathsf{ct}_v) \circ \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_v \end{pmatrix} \right).$$

We remind the readers that the BLS example above is simply a special case of this general framework with a single group element signature (i.e., $v = 1$) and a single pairing equation verification step (i.e., $u = 1$).

2.2 Silent Threshold Signature with Linear Verification

Now that we have established that one can build a signature-based witness encryption scheme for any signature scheme with a linear verification, our next objective is to build a silent threshold signature that comes with a linear verification. At this point, it is helpful to first recall the silent threshold signature scheme from [GJM+24].

Overview of hinTS [GJM+24]. hinTS is a silent threshold signature scheme based on BLS signature. During the silent setup, each party will sample an independent BLS key pair $\{\mathsf{sk}_i, \mathsf{pk} = g^{\mathsf{sk}_i}\}$. In the online phase, given a tag to sign, parties simply sign it using the BLS signature. Now, suppose a subset $B \subseteq [n]$ of parties have signed tag, the aggregator will proceed to aggregate the partial signatures as follows. It first shall aggregate the partial signatures $\{\sigma_i\}_{i \in B}$ and public keys $\{\mathsf{pk}_i\}_{i \in B}$ as σ^* and aPK. Furthermore, to ensure unforgeability, the aggregator must prove the honest aggregation of aPK. In particular, the aggregator needs to commit to the vector B[10] and generating two corresponding succinct proofs π_1, π_2.

1. π_1 proves that aPK is the honest aggregation of public keys for parties in B, i.e.,

$$\mathsf{aPK} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n) \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}.$$

2. π_2 proves that the committed subset B is an authorized set, i.e., $|B| \geqslant t$.

The final signature consists of the aggregated public key aPK, the aggregated signature σ^*, the commitment $\mathsf{Com}(B)$, and the two succinct proofs π_1, π_2. Correspondingly, the verification needs to verify the proofs π_1 and π_2 and verify the aggregated signature σ under the aggregated public key aPK.

Recall that our objective is to have a silent threshold signature with linear verification. Is the hinTS verification entirely linear? First, verifying σ^* under aPK is identical to BLS verification, i.e.,

$$g \circ \sigma^* \stackrel{?}{=} \mathsf{aPK} \circ \mathsf{RO}(\mathsf{tag}).$$

[10] For a set $B \subseteq [n]$, we also think of B as an indicator vector $(b_1, \ldots, b_n) \in \{0,1\}^n$, i.e., $b_i = 1$ if and only if $i \in B$.

Therefore, this part of the verification is linear. To answer whether verifying π_1 and π_2 is linear or not, we need to delve deeper into how $\mathsf{Com}(B), \pi_1, \pi_2$ are generated, which we explain next.

Polynomial Commitment as Vector Commitment. Similar to pairing-based SNARKs, hinTS use KZG polynomial commitment [KZG10] as a succinct vector commitment scheme. That is, for any vector $B = (b_1, b_2, \ldots, b_n)$, it is equivalently treated as a polynomial $B(x)$.[11] The commitment of B is simply the polynomial commitment of $B(x)$, i.e., $[B(\tau)]$,[12] where τ is the trapdoor in the KZG CRS $[\tau], [\tau^2], \ldots, [\tau^n]$.

The Proof π_1. For proving that aPK is the inner product between the vector of public keys and B, hinTS relies on the following polynomial identity (known as generalized sumcheck)

$$\mathsf{SK}(x) \cdot B(x) = \mathsf{aSK} + Q_x(x) \cdot x + Q_Z(x) \cdot Z(x),^{13}$$

where $\deg(Q_x)$ is required to be $\leqslant n-2$. We refer the readers to Lemma 1 for technical details. Given this polynomial identity, the proof consists of the polynomial commitment of $Q_x(x)$ and $Q_Z(x)$, i.e., $\pi_2 = ([Q_x(\tau)], [Q_Z(\tau)])$. Verifying π_2 involves checking the polynomial identity through pairing as

$$[\mathsf{SK}(\tau)] \circ [B(\tau)] \stackrel{?}{=} \mathsf{aPK} \circ [1] + [Q_x(\tau)] \circ [\tau] + [Q_Z(\tau)] \circ [Z(\tau)]. \tag{2}$$

Essentially, this verification step checks that the polynomial identity holds at the random location $x = \tau$. For our purpose, we crucially note that this verification step is linear. In particular, in Equation 2, the group elements from the signature are highlighted and the rest are *public group elements*. Before we move on to π_2, we make a few remarks.

1. Although $[\mathsf{SK}(\tau)]$ is public information, computing this group element involves terms the verifier cannot compute, e.g., $[\mathsf{sk}_1 \cdot \tau]$. Therefore, this scheme crucially relies on each party (holding secret key sk) to also publish $[\mathsf{sk} \cdot \tau], [\mathsf{sk} \cdot \tau^2], \ldots$[14] Similarly, the aggregator also relies on these additional terms to compute $[Q_x(\tau)]$ and $[Q_Z(\tau)]$.

2. As observed by [GJM+24], this proof π_1 is only *weakly sound* in the following sense. If some $[B(\tau)], \mathsf{aPK}, [Q_x(\tau)], [Q_Z(\tau)]$ passes Equation 2 (for instance, the honest generated proof), the adversary can easily produce other tuples of elements, which will also pass Equation 2. For instance,

$$[B(\tau)], \mathsf{aPK} + \tau, [Q_x(\tau) - 1], [Q_Z(\tau)].$$

   However, this is not an issue in terms of unforgeability since the adversary cannot produce a valid signature σ for the (maliciously computed) aggregated public key (e.g., $\mathsf{aPK} + \tau$ in the above example).

3. We are omitting the fact that one needs to check the degree of Q_x is $\leqslant n-2$. As we will see shortly, the degree check is again a linear verification.

The Proof π_2. So far, the verification of hinTS is entirely linear. However, verifying π_2 turns out to be tricky. In particular, hinTS checks $B(x)$ is authorized by checking (1) B is a binary vector, (2) the inner product between B and $(1, 1, \ldots, 1)$ is t.[15] While the second condition can be proven similarly as π_1,

[11] For technical reasons, we treat $(b_1, b_2, \ldots, b_n)$ as the evaluation form of the polynomial instead of the coefficient form.
[12] For $x \in \mathbb{F}$, we use $[x]$ to denote group element $x \cdot g$, where g is the generator of the group. Refer to Section 3.1.
[13] Here, $\mathsf{SK}(x)$ is the polynomial defined by the vector $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n)$ and $Z(x)$ is the (public) vanishing polynomial.
[14] These are referred to as *hints*, which is why the scheme is coined as hinTS.
[15] By changing $(1, 1, \ldots, 1)$ to some weighted vector $(w_1, \ldots, w_n)$, one naturally constructs a weighted threshold signature.

which supports linear verification, proving (1) turns out to be problematic. In particular, one typically uses the polynomial identity

$$B(x) \cdot (1 - B(x)) = Q(x) \cdot Z(x)$$

to prove that B is binary. As highlighted, this is a *degree-2* check (as in one needs to pair $[B(\tau)]$ with itself), which we do not know how to build a witness encryption for. Moreover, it seems inherent that one needs to check if B is binary; otherwise, the adversary may use $B = (t, 0, 0, \ldots, 0)$ to prove that B is authorized even though B contains only one party. This bottleneck raises the following technical question:

How can we prove that B is authorized using only linear verification?

Degree-check to the rescue. Our key observation for addressing this technical challenge is the following. Even if B is not a binary vector, as long as this vector B has $\geqslant t$ *non-zero coordinates*, $\mathsf{aPK} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n) \cdot (b_1, \ldots, b_n)^\intercal$ will be the aggregation of *sufficiently many* (i.e., $\geqslant t$) public keys, which by the security of BLS multisignature, is unforgeable if one does not have $\geqslant t$ partial signatures.

Now, checking that B has $\geqslant t$ non-zero coordinates could actually be done by a linear check. In particular, one may check this by running a degree check on $B(x)$. Intuitively, if a *non-zero* polynomial has degree $\leqslant n - t$, its evaluations will have $\leqslant n - t$ zeros and, hence, $\geqslant t$ non-zeros. Now, suppose the CRS is $[\tau], [\tau^2], \ldots, [\tau^n]$, checking if a committed polynomial $B(x)$ has degree $\leqslant n - t$ simply means, asking the prover to also commit to $\widehat{B}(x) = B(x) \cdot x^t$ and check if

$$[B(\tau)] \circ [\tau^t] \stackrel{?}{=} [\widehat{B}(\tau)] \circ [1].$$

Crucially for us, note that this is again a linear check.

Are we done? One subtlety is that we do require the committed polynomial $B(x)$ to be non-zero, i.e., $[B(\tau)] \neq [0]$. This check is actually non-linear. However, this can be simply fixed by introducing a dummy party P_0 and always requiring $\mathsf{P}_0 \in B$. Proving $\mathsf{P}_0 \in B$ introduces another KZG opening proof, whose verification again conforms to a linear check.

2.3 Putting it together

We are now ready to put everything together to build our silent threshold encryption. First, the silent threshold signature scheme with a linear verification is summarized as follows. During silent setup phase, each party independently samples sk, pk and publishes pk together with the hints $[\mathsf{sk} \cdot \tau], [\mathsf{sk} \cdot \tau^2], \ldots$ Given a random group elements $[\gamma]$ to sign,[16] parties partially sign it as $[\gamma \cdot \mathsf{sk}]$. The aggregator aggregates these partial signatures into

$$\mathsf{aPK},\ \sigma^*,\ [B(\tau)],\ [Q_x(\tau)],\ [Q_Z(\tau)],\ [\hat{Q}_x(\tau)],\ [\hat{B}(\tau)],\ [Q_0(\tau)]$$

which should satisfy the linear verification through the following five pairing equations.

$$[\mathsf{SK}(\tau)]_1 \circ [B(\tau)]_2 = [1]_2 \circ \mathsf{aPK} + [Z(\tau)]_2 \circ [Q_Z(\tau)]_1 + [\tau]_2 \circ [Q_x(\tau)]_1 \qquad \text{(Sumcheck)}$$

$$[\tau]_2 \circ [Q_x(\tau)]_1 = [1]_2 \circ [\hat{Q}_x(\tau)]_1 \qquad \text{(Degree-check)}$$

$$[\gamma]_2 \circ \mathsf{aPK} = [1]_1 \circ \sigma^* \qquad \text{(Signature Verification)}$$

$$[\tau^t]_1 \circ [B(\tau)]_2 = [1]_2 \circ [\hat{B}(\tau)]_1 \qquad \text{(Degree-check)}$$

$$[1]_1 \circ [B(\tau)]_2 = [\tau - 1]_2 \circ [Q_0(\tau)]_1 + 1 \qquad \text{(Dummy Party)}$$

[16] Looking ahead, this element is sampled by the encryptor in the encryption scheme. Therefore, it is not necessary to use a random oracle to sample a random group element as in the signature scheme.

As a sanity check, note that without introducing the dummy party, the linear system is trivially satisfiable (since all zero is a trivial solution), which means the corresponding signature scheme is trivially forgeable. Given this silent threshold signature scheme, one compiles it into an encryption scheme just as described in Section 2.1. In particular, our STS is a signature with $u = 8$ group elements and $v = 5$ pairing check verification.

This sums up our construction on a high level. We next discuss a few more points.

Security. We prove the security of our scheme in the generic group model (GGM) [Sho97, Mau05]. Intuitively, the security of the encryption scheme reduces to the unforgeability of the signature scheme. In GGM, the adversary can distinguish a ciphertext from a random group element if and only if it can derive this element by generic operations. By careful argument, this means that the adversary must be able to find $(\sigma_1, \ldots, \sigma_v)$ that satisfies Equation 1, which gives us a forgery. One may argue that this cannot happen using similar reasoning as hinTS [GJM+24].

Efficiency. In our framework of constructing STE from STS: 1. the encryption key (of STE) is the verification key (of STS); 2. the partial decryption (of STE) is the partial signing (of STS); 3. decryption aggregation (of STE) is the partial signature aggregation (of STS), plus a few more constant-time operations (corresponding to the SWE). Therefore, our scheme inherits the efficiency of the STS scheme of the (modified) hinTS [GJM+24]. We provide implementation and evaluation details in Section 7.

Extensions. Our basic and CPA-secure scheme is described in Section 5. We then proceed to present some extensions to the basic scheme. First we show that it readily enjoys multiverse setting. Then we show a simple extension that has CCA2 security. We show that a straightforward variant of our scheme for $t = 1$ serves as a direct Flexible Broadcast Encryption construction with constant-sized public keys. Finally we discuss how to achieve STE with the highly desirable security properties of Post-Compromise and Forward Security. For the former we argue that it is a property that we get for free from the Silent Setup setting. For the latter we discuss how one can rely on a Forward Secure STS (seen as a variant of hinTS) to build a Forward Secure STE.

These extensions are discussed in Section 6.

3 Preliminaries

Notation. Throughout this work, we use λ for the security parameter and $\mathsf{negl}(\lambda)$ for a negligible function, i.e., a function that is less than $1/f(\lambda)$ for *any* polynomial f. We use $[n]$ to denote the set $\{1, \ldots, n\}$ and $[a, b]$ (for $a, b \in \mathbb{Z}$ and $a < b$) the set $\{a, a + 1, \ldots, b\}$.

3.1 Bilinear Groups

A Bilinear Group $\mathcal{BG}$, generated as $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathcal{BG}(1^\lambda)$, is specified by three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ (the first two we call 'source groups' and the third 'target group') of prime order $p = 2^{\Theta(\lambda)}$, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that we call 'pairing' and one random generator g_1, g_2 for each group. We use the *implicit notation*, i.e., $[x]_s := x \cdot g_s$ and more generally $[\boldsymbol{A}]_s$ represents a matrix of the corresponding group elements, for $s \in \{1, 2, t\}$. Also, we denote the group operation additively, $[x]_s + [y]_s = [x + y]_s$, for $s \in \{1, 2, t\}$. By $[x]_1 \circ [y]_2 = [y]_2 \circ [x]_1 = [x \cdot y]_T$, we denote the pairing $e([x]_1, [y]_2)$. We note that the way it is defined '$\circ$' is commutative, for instance $([a]_1, [b]_2)^\intercal \circ ([c]_2, [d]_1)$ is well-defined and gives the outcome $[ac + bd]_T$.

All the algorithms of our constructions implicitly take as input a Bilinear Group generated by $\mathcal{BG}(\lambda)$, even if it is not explicitly stated.

For completeness, we include a definition of BLS signature [BLS01] below.

Definition 1 (BLS Signature). *Let* $\mathsf{RO} : \{0,1\}^* \to \mathbb{G}_2$ *be a random oracle. The BLS signature consists of the following algorithms.*

- blsgen*: It samples a random* $\mathsf{sk} \leftarrow \mathbb{F}$ *and output a public/secret key pair as* $(\mathsf{pk} = [\mathsf{sk}]_1, \mathsf{sk})$.
- $\mathsf{blssign}(\mathsf{msg})$*: It signs as* $\sigma = \mathsf{sk} \cdot \mathcal{H}(\mathsf{msg})$.
- $\mathsf{blsver}(\mathsf{pk}, \mathsf{msg}, \sigma)$*: It verifies the validity of the signature by* $\mathsf{pk} \circ \mathsf{RO}(\mathsf{msg}) \stackrel{?}{=} [1]_1 \circ \sigma$.

3.2 Polynomials over $\mathbb{Z}_p$

Throughout the paper, we use the following notations for polynomials over the field $\mathbb{Z}_p$, defined by the bilinear group. Let $\omega \in \mathbb{Z}_p$ be an ℓ-th primitive root of unity, i.e. $\omega^\ell = 1$ over $\mathbb{Z}_p$. ω generates the multiplicative subgroup of roots of unity $\mathbb{H} = \{\omega, \omega^2, \ldots, \omega^\ell\}$, with $|\mathbb{H}| = \ell$. Let $L_1(x), L_2(x), \ldots, L_\ell(x)$ denote the Lagrange basis polynomial. That is, L_i is the unique degree-$(\ell-1)$ polynomial defined by: $L_i(\omega^j)$ is 1 when $i = j$ and 0 when $i \neq j$. Let $Z(x) = \prod_{i=1}^{\ell}(x - \omega^i)$ be the vanishing polynomial on $\mathbb{H}$. Since $\mathbb{H}$ is a multiplicative subgroup, $Z(x) = x^\ell - 1$ and $L_i(x) = \frac{\omega^i}{\ell} \cdot \frac{x^\ell - 1}{x - \omega^i}$. Note that $L_i(0) = \frac{1}{\ell}$. Sometimes, we will refer to ω^ℓ as ω_0 and $L_0 = L_\ell$; since $\mathbb{H}$ is cyclic, they are equivalent.

3.3 Generic Group Model

Generic (Bilinear) Group model (GGM). Our security proof is based on the Generic Group Model [Sho97, Mau05]. A 'generic' adversary does not have concrete representations of the elements of the group and can only use generic group operations. This model captures the possible 'algebraic' attacks that an adversary can perform.

In particular, we follow the Maurer's GGM [Mau05], which is extended to Bilinear Groups by [BBG05]. The adversary in GGM makes oracle queries for each generic group operation she wishes to perform and receives a handle for the resulting group element, instead of the actual element itself. We call the party that answers the queries the 'challenger'. The challenger keeps three lists $\boldsymbol{L}_1, \boldsymbol{L}_2, \boldsymbol{L}_T$ of all group elements resulted from the queries of the adversary together with their corresponding handles.

A standard GGM technique in security proofs is the 'symbolic' equivalence. We call 'symbolic' experiment (and symbolic group representation, respectively) the model where the challenger is storing polynomials instead of group elements and performs polynomial operations instead of group operations. The formal variables of the polynomials are the initial elements that the adversary received. For example, a generic adversary to the discrete logarithm problem is initially receiving $[1], [x]$; thus, the formal variables are $1, X$, and then can perform any generic group operation which is going to be symbolically performed by the challenger with the corresponding polynomials in $\mathbb{Z}_p[1, X]$.

Master Theorem. We recall the 'Master Theorem' [BBG05, Boy08] that simplifies the proofs of the hardness of decisional problems.

Theorem 1 (Master theorem [BBG05, Boy08]). *Let* $\boldsymbol{L}_1 \in \mathbb{Z}_p[X_1, \ldots, X_n]^{\nu_1}$, $\boldsymbol{L}_2 \in \mathbb{Z}_p[X_1, \ldots, X_n]^{\nu_2}$, $\boldsymbol{L}_3 \in \mathbb{Z}_p[X_1, \ldots, X_n]^{\nu_T}$ *be three lists*[17] *of* n*-variate polynomials over* $\mathbb{Z}_p$ *of maximum degree* $d_{\boldsymbol{L}_1}, d_{\boldsymbol{L}_2}, d_{\boldsymbol{L}_T}$*, respectively. Let* $f \in \mathbb{Z}_p[X_1, \ldots, X_n]$ *be an* n*-variate polynomial of degree* d_f *and denote* $d = \max\{d_{\boldsymbol{L}_1} +$

[17] Throughout this section, we will abuse the notation sometimes, treating lists as vectors.

$d_{\boldsymbol{L}_2}, d_{\boldsymbol{L}_T}, d_f\}$, $\nu = \nu_1 + \nu_2 + \nu_3$. *If* f *is* independent *of* $(\boldsymbol{L}_1, \boldsymbol{L}_2, \boldsymbol{L}_T)$, *then for any generic adversary* $\mathcal{A}$ *that makes at most* q *group oracle queries:*

$$\left|\Pr\left[\mathcal{A}\begin{pmatrix} p, \mathsf{h}_1[\boldsymbol{L}_1(\boldsymbol{x})], \\ \mathsf{h}_2[\boldsymbol{L}_2(\boldsymbol{x})], \mathsf{h}_T[f(\boldsymbol{x})]\end{pmatrix} = 1\right] - \Pr\left[\mathcal{A}\begin{pmatrix} p, \mathsf{h}_1[\boldsymbol{L}_1(\boldsymbol{x})], \\ \mathsf{h}_2[\boldsymbol{L}_2(\boldsymbol{x})], \mathsf{h}_T[r]\end{pmatrix} = 1\right]\right| \leqslant \frac{(q+\nu+2)^2 \cdot d}{2p},$$

where $\mathsf{h}_1, \mathsf{h}_2, \mathsf{h}_T$ *denote the corresponding handles, and the probabilities are taken over the choices of* $\boldsymbol{x} \leftarrow\!\!\$\ (\mathbb{Z}_p)^n$ *and* $r \leftarrow\!\!\$\ \mathbb{Z}_p$.

We have yet to specify what the f-(in)depenence of $\boldsymbol{L} = (\boldsymbol{L}_1, \boldsymbol{L}_2, \boldsymbol{L}_T)$ means. First, define the *completion* [BFF+14] of $\boldsymbol{L}$ as

$$\boldsymbol{C}(\boldsymbol{L}) := \{\boldsymbol{L}_1 \otimes \boldsymbol{L}_2\} \cup \boldsymbol{L}_T.$$

Intuitively $\{\boldsymbol{L}_1 \otimes \boldsymbol{L}_2\}$ are all the elements in $\mathbb{G}_T$ that can be computed using pairings. Given this, f-(in)dependence is defined as follows.

Definition 2. *Let the lists of polynomials* $\boldsymbol{L}_1, \boldsymbol{L}_2, \boldsymbol{L}_T$ *with elements in* $\mathbb{Z}_p[X_1, \ldots, X_n]$, *the polynomial* $f \in \mathbb{Z}_p[X_1, \ldots, X_n]$ *and* $\boldsymbol{C}(\boldsymbol{L}) = \{g_1(X_1, \ldots, X_n), \ldots, g_D(X_1, \ldots, X_n)\}$. *We say that* f *is dependent on* $\boldsymbol{L} = (\boldsymbol{L}_1, \boldsymbol{L}_2, \boldsymbol{L}_T)$ *if there exist coefficients* $\kappa_i \in \mathbb{Z}_p$ *such that:*

$$f(X_1, \ldots, X_n) = \sum_{i=1}^{D} \kappa_i \cdot g_i(X_1, \ldots, X_D).$$

Otherwise, we say that f *is independent of* $\boldsymbol{L}$.

3.4 Univariate Sumcheck

Our construction relies on a univariate sumcheck protocol [BCR+19, RZ21], slightly modified to work for inner products [CNR+22]. In particular, we use the following lemma.

Lemma 1 (Univariate Sumcheck [BCR+19, RZ21])**.** *Let* $A(x) = \sum_{i=1}^{|\mathbb{H}|} a_i \cdot L_i(x)$, $B(x) = \sum_{i=1}^{|\mathbb{H}|} b_i \cdot L_i(x)$. *It holds that*

$$A(x) \cdot B(x) = \frac{\sum_i a_i b_i}{|\mathbb{H}|} + Q_x(x) \cdot x + Q_Z(x) \cdot Z(x),$$

where both Q_x *and* Q_Z *are polynomials with degree* $\leqslant |\mathbb{H}| - 2$ *defined as*

$$Q_x(x) = \sum_i a_i b_i \frac{L_i(x) - L_i(0)}{x},$$

$$Q_Z(x) = \sum_i a_i b_i \frac{L_i^2(x) - L_i(x)}{Z(x)} + \sum_{i \neq j} a_i b_j \frac{L_i(x) L_j(x)}{Z(x)}$$

We note that the original sumcheck is concretely stated for $b_i = 1$. In our case, we treat general inner products, which is a straightforward generalization (see [CNR+22]).

4 Defining Silent Threshold Encryption

This section formally defines the primitive: *silent threshold encryption* (STE). Our formal definition below is inspired by the silent threshold signature definition of [GJM+24]. In particular, parties will publish some "hints" together with their public key in a *silent* manner. Given all the hints, a public algorithm will verify the validity of the hints. Furthermore, a *succinct* encryption key will be deterministically computed from the hints.

Definition 3 (STE)**.** *A Silent Threshold Encryption consists of a tuple of algorithms* $\Sigma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{HintGen}, \mathsf{Preprocess}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{PartVerify}, \mathsf{DecAggr})$ *with the following syntax.*

- $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, M)$*: On input the security parameter* λ *and an upper bound* M *on the maximum number of users, the* Setup *algorithm outputs a common reference string* CRS*.*
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$*: On input the security parameter* λ*, the* KGen *algorithm outputs a public/secret key pair* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{hint}_i \leftarrow \mathsf{HintGen}(\mathsf{CRS}, \mathsf{sk}, M, i)$*: On input the* CRS*, the secret key* sk*, the number of parties* M*, and a position* $i \in [M]$*, the* HintGen *algorithm outputs a hint* hint_i*.*
- $(\mathsf{ak}, \mathsf{ek}) \leftarrow \mathsf{Preprocess}(\mathsf{CRS}, \mathcal{U}, \{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in\mathcal{U}})$*: On input the* CRS*, a universe* $\mathcal{U} \subseteq [M]$*, all pairs* $\{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in\mathcal{U}}$*, the* Preprocess *algorithm computes an aggregation key* ak *and a encryption key* ek*.*
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{msg}, t)$: *On input an encryption key* ek*, a message* msg *and a threshold* t*, it outputs a ciphertext* ct*.*
- $\sigma \leftarrow \mathsf{PartDec}(\mathsf{sk}, \mathsf{ct})$: *On input a secret key* sk*, and a ciphertext* ct*,* PartDec *algorithm outputs a partial decryption* σ*.*
- $1/0 \leftarrow \mathsf{PartVerify}(\mathsf{ct}, \sigma, \mathsf{pk})$: *On input a ciphertext* msg*, a partial decryption* σ*, and a public key* pk*, it returns* 1 *if and only if the partial decryption verifies.*
- $\mathsf{msg} \leftarrow \mathsf{DecAggr}(\mathsf{CRS}, \mathsf{ak}, \mathsf{ct}, \{\sigma_i\}_{i\in S})$: *On input the* CRS*, an aggregation key* ak*, and a set of partial decryptions* $\{\sigma_i\}_{i\in S}$*, the* DecAggr *algorithm outputs a message* msg*.*

Moreover, STE must have the following efficiency *requirements:*

- *The aggregation key* ak *and the ciphertext* ct *should be constant size.*
- *Partial decryption should only take constant time.*

Remark 1 (Silent Setup)**.** *We note that* HintGen *does not take other parties' public keys* pk_i*'s as input. It solely depends on the CRS and, hence, parties can publish* $(\mathsf{pk}_i, \mathsf{hint}_i)$ *in* one shot*. In other words,* $(\mathsf{pk}_i, \mathsf{hint}_i)$ *can be viewed as the (extended) public key of party* i*.*

Remark 2 (Preprocessing)**.** *The preprocessing algorithm is only decoupled from encryption and decryption aggregation for efficiency reasons. In terms of functionality and security, it could be equivalently embedded in the* Enc *and* DecAggr *algorithms.*

This decoupling is helpful in the threshold encryption setting, where a universe $\mathcal{U}$ *is generated once, and an encryptor will encrypt with respect to* $\mathcal{U}$ *repetitively. In this way, the preprocessing cost is* amortized*. In other applications, such as threshold broadcast encryption (see Section 1.2), one may equivalently embed processing inside encryption and decryption aggregation.*

Due to the complexity of the primitive, we define correctness through a game between a challenger and an adversary. Note that the adversary is computationally unbounded, and the correctness is perfect. Intuitively, we allow the adversary to choose the universe $\mathcal{U}$ and control any number of users in $\mathcal{U}$. Furthermore, the adversary can output the partial decryptions of any party it controls. It is important to note that correctness should hold even for *maliciously generated* public keys, hints, and partial decryptions of users controlled by the adversary. The formal definition of correctness can be found below.

1. The challenger runs $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, M)$ and gives CRS to $\mathcal{A}$.
2. The adversary picks a universe $\mathcal{U}^*$ and a subset Cor of corrupt parties such that $\mathsf{Cor} \subseteq \mathcal{U}^*$.
3. For all $i \in \mathcal{U}^* \setminus \mathsf{Cor}$, the public key and hint are sampled honestly $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(1^\lambda)$ and $\mathsf{hint}_i = \mathsf{HintGen}(\mathsf{CRS}, \mathsf{sk}_i, M, i)$.
4. For all $i \in \mathsf{Cor}$, the adversary returns a (potentially maliciously generated) public key pk_i and hint hint_i to the challenger.
5. The challenger invokes the preprocessing as $(\mathsf{ek}, \mathsf{ak}) \leftarrow \mathsf{Preprocess}(\mathsf{CRS}, \mathcal{U}^*, \{\mathsf{hint}_i, \mathsf{pk}_i\}_{i \in \mathcal{U}^*})$ and the outputs are given to $\mathcal{A}$.
6. The adversary is given access to the partial decryption oracle of the honest parties, and it picks a message msg and a threshold t.
7. The honest ciphertext is generated $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{msg}, t)$.
8. The adversary prepares the partial decryptions $\{\sigma_i\}_{i \in S_1}$ for some subset of malicious parties $S_1 \subseteq \mathsf{Cor}$. Let $S_1' \subseteq S_1$ be the subset of maliciously generated partial decryptions that *verifies* under $\mathsf{PartVerify}$.
9. The adversary may also request a subset of honest parties $S_2 \subseteq \mathcal{U}^* \setminus \mathsf{Cor}$ for partial decryptions, which are returned by computing $\sigma_i \leftarrow \mathsf{PartDec}(\mathsf{sk}_i, \mathsf{ct})$.
10. If $|S_1' \cup S_2| \geqslant t$ (i.e., there are sufficiently many *verified* partial decryptions), the challenger computes the aggregated decryption as $\mathsf{msg}' \leftarrow \mathsf{DecAggr}(\mathsf{CRS}, \mathsf{ak}, \mathsf{ct}, \{\sigma_i\}_{i \in S})$, where $S = S_1' \cup S_2$.
11. The output of this game is 0 if $\mathsf{msg} \neq \mathsf{msg}'$.

Figure 1: Correctness Game

Definition 4 (Correctness)**.** *The STS scheme Σ satisfies correctness if, for any unbounded adversary $\mathcal{A}$ and any $M = \mathsf{poly}(\lambda)$, the output of the correctness game defined in Figure 1 is 0 with probability 0.*

For the security of STE, we naturally define a semantic security game between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$. Again, the adversary chooses the target universe $\mathcal{U}^*$ and corrupts any subset of parties, denoted by Cor, in $\mathcal{U}^*$. Later, the adversary also chooses the threshold t (adaptively). For the security to be meaningful, we demand that $t > |\mathsf{Cor}|$; otherwise, $\mathcal{A}$ can trivially win the game.

Definition 5 (Semantic Security)**.** *The STE scheme Σ satisfies semantic security if, for every $M = \mathsf{poly}(\lambda)$ and any adversary PPT $\mathcal{A}$, the output of the game in Figure 2 is 1 with probability $\leqslant 1/2 + \mathsf{negl}(\lambda)$.*

1. *The challenger runs* $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, M)$ *and gives* CRS *to* $\mathcal{A}$.
2. *The adversary picks* $\mathcal{U}^*$ *and a subset of parties to corrupt* $\mathsf{Cor} \leftarrow \mathcal{A}(\mathsf{CRS})$.
3. *For all honest parties* $i \in \mathcal{U}^* \setminus \mathsf{Cor}$*, the public key and hint are sampled honestly by the challenger* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(1^\lambda)$ *and* $\mathsf{hint}_i = \mathsf{HintGen}(\mathsf{CRS}, \mathsf{sk}_i, M, i)$ *and are sent to* $\mathcal{A}$.
4. *For all* $i \in \mathsf{Cor}$*, the adversary picks a public key* pk_i *and the corresponding hint* hint_i.
5. *The challenger invokes the preprocessing as* $(\mathsf{ek}, \mathsf{ak}) \leftarrow \mathsf{Preprocess}(\mathsf{CRS}, \mathcal{U}^*, \{\mathsf{hint}_i, \mathsf{pk}_i\}_{i \in \mathcal{U}^*})$ *and the output are given to* $\mathcal{A}$.
6. *The adversary picks messages* msg_0, msg_1, *and a threshold* t.
7. *The challenger picks a bit* $b \leftarrow \{0,1\}$ *and generates a ciphertext* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{msg}_b, t)$.
8. *The adversary outputs a bit* b' *and wins the semantic security game if* $t > |\mathsf{Cor}|$ *and* $b' = b$*, in which case, the output of the game is 1.*

Figure 2: Security Game

Remark 3. *The adversary* $\mathcal{A}$ *can also corrupt parties outside the target universe* $\mathcal{U}^*$*. However, they do not play any role as they do not participate in the decryption committee, i.e., their public keys are not taken as input on* Enc *or* DecAggr*. For correctness and security, we can ignore them; equivalently, one may consider all parties outside* $\mathcal{U}^*$ *corrupted.*

5 Our Silent Threshold Encryption Construction

This section presents our core contribution: our construction of a silent threshold encryption scheme. First, in Section 5.1, we show the description of the construction. Then, Section 5.2 presents the analysis of the scheme: its (asymptotic) efficiency, correctness, and security proof.

5.1 Construction

Here, we describe formally our silent threshold encryption scheme. For an intuitive description of the construction, we refer to Section 2. As noted there, the core of the construction is a (witness) encryption with respect to a matrix $\boldsymbol{A}$ representing the verification of a threshold signature. To ease the presentation, we explain here where this matrix comes from. Consider the following five pairing equations[18] (recall from Section 3 that '$\circ$' is commutative).

1. $[\mathsf{SK}(\tau)]_1 \circ [B(\tau)]_2 = [1]_2 \circ \mathsf{aPK} + [Z(\tau)]_2 \circ [Q_Z(\tau)]_1 + [\tau]_2 \circ [Q_x(\tau)]_1$
2. $[\tau]_2 \circ [Q_x(\tau)]_1 = [1]_2 \circ [\hat{Q}_x(\tau)]_1$
3. $[\gamma]_2 \circ \mathsf{aPK} = [1]_1 \circ \sigma^*$
4. $[\tau^t]_1 \circ [B(\tau)]_2 = [1]_2 \circ [\hat{B}(\tau)]_1$
5. $[1]_1 \circ [B(\tau)]_2 = [\tau - 1]_2 \circ [Q_0(\tau)]_1 + 1$

[18] The shaded elements are from the signature. The rest are public group elements.

where $\mathsf{SK}(X) = \sum_{i=1}^{M} \mathsf{sk}_i L_i(X)$. This, essentially, yields a Silent Threshold Signature verification. In particular, this is a variant of hinTS [GJM+24].
Intuitively, the first equation is for proving the honest aggregation of aPK by the univariate sumcheck (Lemma 1). The second and fourth equation is for the degree check on Q_x and B. The third equation verifies the aggregated signature σ (for a random tag $[\gamma]_2$) under the aggregated public key aPK. Finally, the fifth equation is for checking that a dummy party is always included in B, so that setting everything to $[0]$ does not give a valid solution.
In matrix form, this can be written as $\boldsymbol{A} \circ \boldsymbol{w} = \boldsymbol{b}$, where

$$\boldsymbol{w} = \left([B(\tau)]_2, -\mathsf{aPK}, [-Q_Z(\tau)]_1, [Q_x(\tau)]_1, [\hat{Q}_x(\tau)]_1, \sigma^*, [\hat{B}(\tau)]_1, [Q_0(\tau)]_1\right)^\top$$

is the aggregated signature. In conclusion, the matrix $\boldsymbol{A}$ comes from the above linear verification. The only difference is that, we can replace $\mathsf{RO}(\mathsf{tag})$ with a random element $[\gamma]_2$ during the encryption.
Henceforth, we will consider that $M+1$ is a power of 2 and we set the subgroup $\mathbb{H} = \{\omega^0, \omega^1, \ldots, \omega^M\}$ of roots of unity to be such that $|\mathbb{H}| = M+1$. M is the maximum number of decryptors participating in the system. We reserve an artificial position 0 that always has $\mathsf{sk}_0 = 1$. No actual user is allowed to use 0 as her index. We consider that the artificial user 0 is always in the set of the universe $\mathcal{U}$.

Construction 1. *We present below a formal description of our silent threshold encryption scheme:*

- $\mathsf{Setup}(1^\lambda)$*: Sample* $\tau \leftarrow\!\!\$\ \mathbb{Z}_p$ *and output:* [19]

$$\mathsf{CRS} = \left([\tau^1]_1, \ldots, [\tau^{M+1}]_1, [\tau^1]_2, \ldots, [\tau^{M+1}]_2\right).$$

- $\mathsf{KGen}(1^\lambda)$*: Sample* $x \leftarrow\!\!\$\ \mathbb{Z}_p^*$ *and output* $\mathsf{pk} = [x]_1$, $\mathsf{sk} = x$.

- $\mathsf{HintGen}(\mathsf{CRS}, \mathsf{sk}_i, i, M)$*: output*

$$\mathsf{hint}_i = \Bigg([\mathsf{sk}_i L_i(\tau)]_1, [\mathsf{sk}_i\big(L_i(\tau) - L_i(0)\big)]_1, \left[\mathsf{sk}_i \frac{L_i^2(\tau) - L_i(\tau)}{Z(\tau)}\right]_1, \left[\mathsf{sk}_i \frac{L_i(\tau) - L_i(0)}{\tau}\right]_1, \left\{\left[\mathsf{sk}_i \frac{L_i(\tau)L_j(\tau)}{Z(\tau)}\right]_1\right\}_{j\in[0,M],j\neq i} \Bigg).$$

- $\mathsf{Preprocess}(\mathsf{CRS}, \mathcal{U}, \{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in\mathcal{U}})$*: Verify the validity of each hint: for each* $i \in \mathcal{U}$ *run* $\mathsf{isValid}(\mathsf{CRS}, \mathsf{hint}_i, \mathsf{pk}_i)$ *(*isValid *is defined below) and let* $V \subseteq \mathcal{U}$ *be the set of the indices with valid hints. Set* $\mathsf{sk}_0 = 1$ *and* $\mathsf{sk}_i = 0$ *for each* $i \notin \mathcal{U}$ *outside the universe. Implicitly, set* $\mathsf{sk}_i = 0$ *for each* $i \notin V$ *and for each* $i \in [M] \setminus \mathcal{U}$ *outside the universe. Output*

$$\mathsf{ak} = \Bigg(V, \{\mathsf{pk}_i\}_{i\in V\cup\{0\}}, \left\{[\mathsf{sk}_i\big(L_i(\tau) - L_i(0)\big)]_1\right\}_{i\in V\cup\{0\}}, \left\{\left[\mathsf{sk}_i \frac{L_i^2(\tau) - L_i(\tau)}{Z(\tau)}\right]_1\right\}_{i\in V\cup\{0\}}, \left\{\left[\mathsf{sk}_i \frac{L_i(\tau) - L_i(0)}{\tau}\right]_1\right\}_{i\in V\cup\{0\}}, \left\{\left[\sum_{j\in S, j\neq i} \mathsf{sk}_j \frac{L_i(\tau)L_j(\tau)}{Z(\tau)}\right]_1\right\}_{i\in V\cup\{0\}} \Bigg).$$

$$\mathsf{ek} = \left(\left[\sum_{i\in V} \mathsf{sk}_i L_i(\tau)\right]_1, [Z(\tau)]_2\right) := (C, Z)$$

[19] For efficiency, we assume that all algorithms also have direct access to $\{[L_i(\tau)]_{1,2}\}_i$, which can be efficiently computed from $\{[\tau^i]_{1,2}\}$ without knowing τ.

- $\mathsf{Enc}(\mathsf{ek}, \mathsf{msg}, t)$: *Sample* $[\gamma]_2 \leftarrow\!\!\$\ \mathbb{G}_2$, *parse* $\mathsf{ek} := (C, Z)$, *set* $Z_0 = [\tau - \omega^0]_2$ *and set*

$$\boldsymbol{A} = \begin{pmatrix} C & [1]_2 & Z & [\tau]_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & [\tau]_2 & [1]_2 & 0 & 0 & 0 \\ 0 & [\gamma]_2 & 0 & 0 & 0 & [1]_1 & 0 & 0 \\ [\tau^t]_1 & 0 & 0 & 0 & 0 & 0 & [1]_2 & 0 \\ [1]_1 & 0 & 0 & 0 & 0 & 0 & 0 & Z_0 \end{pmatrix}$$
$$\boldsymbol{b} = ([0]_T, [0]_T, [0]_T, [0]_T, [1]_T)^\top .$$

Notice that each column of $\boldsymbol{A}$ *contains elements from the same source group (looking ahead, this is so that the pairing* $\boldsymbol{A} \circ \boldsymbol{w}$ *can be properly performed.)*

Sample a vector $\boldsymbol{s} \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^5$ *and output*

$$\mathsf{ct} = \left([\gamma]_2, \boldsymbol{s}^\top \cdot \boldsymbol{A}, \boldsymbol{s}^\top \cdot \boldsymbol{b} + \mathsf{msg}\right).$$

- $\mathsf{PartDec}(\mathsf{sk}, \mathsf{ct})$: *Parse* $\mathsf{ct} := ([\gamma]_2, \mathbf{ct}_2, \mathsf{ct}_3)$ *and output* $\sigma = \mathsf{sk} \cdot [\gamma]_2$.
- $\mathsf{PartVerify}(\mathsf{ct}, \sigma, \mathsf{pk})$: *Parse* $\mathsf{ct} := ([\gamma]_2, \mathbf{ct}_2, \mathsf{ct}_3)$ *and output* 1 *if and only if* $\mathsf{pk} \circ [\gamma]_2 = [1]_1 \circ \sigma$.
- $\mathsf{DecAggr}(\mathsf{CRS}, \mathsf{ak}, \mathsf{ct}, \{\sigma_i\}_{i \in S})$: *Using* ak *compute the subset of indices with valid hints,* $S_v = S \cap V$. *Then proceed as follows:*

  1. *Compute a polynomial* $B(X)$ *by interpolating* 0 *on all* ω_i *with* $i \notin S_v$ *and* 1 *on* ω^0, *i.e., interpolate* B *as* $\{(\omega^0, 1), (\omega^i, 0)_{i \notin S_v}\}$ *and set*

$$B = \left[\sum_{i=0}^{M} B(\omega^i) L_i(\tau)\right]_2$$

  2. *Set* $\mathsf{aPK} = \frac{1}{M+1}(\sum_{i \in S_v} B(\omega^i)\mathsf{pk}_i + [1]_1)$.
  3. *Compute polynomials* $Q_x(X)$ *and* $Q_Z(X)$ *such that*

$$\mathsf{SK}(X)B(X) = \frac{\mathsf{aSK}}{M+1} + Q_Z(X)Z(X) + Q_x(X)X$$

  where $\mathsf{aSK} = \sum_{i=0}^{M} \mathsf{SK}(\omega^i)B(\omega^i) := \sum_{i \in S_v} \mathsf{sk}_i B(\omega^i) + 1$ *(since, by definition,* B *evaluates to* 0 *outside* S_v *and to* 1 *on* ω^0*).*

  According to Lemma 1 they can be computed as:

$$Q_Z(X) = \sum_{i=0}^{M+1} B(\omega^i) \left(\mathsf{sk}_i \frac{L_i^2(X) - L_i(X)}{Z(X)}\right) + \left(\sum_{i=0}^{M+1} B(\omega^i) \sum_{j=0, j \neq i}^{M+1} \mathsf{sk}_j \frac{L_i(X)L_j(X)}{Z(X)}\right),$$
$$Q_x(X) = \sum_{i=0}^{M+1} B(\omega^i) \left(\mathsf{sk}_i \frac{L_i(X) - L_i(0)}{X}\right).$$

  Then using ak *compute* $Q_Z = [Q_Z(\tau)]_1$.
  4. *Compute* $Q_x = [Q_x(\tau)]_1$.
  5. *Compute* $\hat{Q}_x = [Q_x(\tau) \cdot \tau]_1$

6. *Set* $\sigma^* = \frac{1}{M+1}(\sum_{i \in S_v} B(\omega^i)\sigma_i + \mathsf{ct}_1)$.
7. *Compute* $\hat{B} = [\tau^t B(\tau)]_1$
8. *Compute a KZG evaluation at* ω^0*, i.e., compute* $Q_0(X)$ *such that* $B(X) - 1 = Q_0(X)(X - \omega^0)$ *and compute* $Q_0 = [Q_0(\tau)]_1$.

Set $\boldsymbol{w} = \left(B, -\mathsf{aPK}, -Q_Z, -Q_x, \hat{Q}_x, \sigma^*, -\hat{B}, -Q_0\right)^\top$*, parse* $\mathsf{ct} = (\mathsf{ct}_1, \mathbf{ct}_2, \mathsf{ct}_3)$ *and output:*

$$\mathsf{msg}^* = \mathsf{ct}_3 - \mathbf{ct}_2 \circ \boldsymbol{w}$$

In order to confirm that hint_i is well-formed in the Preprocess algorithm we use an isValid algorithm which we define as follows.

$\mathsf{isValid}(\mathsf{CRS}, \mathsf{hint}_i, \mathsf{pk}_i) \to \{0,1\}$: parses $\mathsf{hint}_i := \left(h_1, h_2, h_3, h_4, \{h_{5,j}\}_{j \in [0,M], j \neq i}\right)$ and output 1 iff it holds that:

1. $h_1 \circ [1]_2 = \mathsf{pk}_i \circ [L_i(\tau)]_2$,
2. $h_2 \circ [1]_2 = \mathsf{pk}_i \circ [(L_i(\tau) - L_i(0))]_2$,
3. $h_3 \circ [1]_2 = \mathsf{pk}_i \circ [\frac{L_i^2(\tau) - L_i(\tau)}{Z(\tau)}]_2$,
4. $h_4 \circ [1]_2 = \mathsf{pk}_i \circ [\frac{L_i(\tau) - L_i(0)}{\tau}]_2$,
5. $h_5 \circ [1]_2 = \mathsf{pk}_i \circ [\frac{L_i(\tau) L_j(\tau)}{Z(\tau)}]_2$, for each $j \in [0, M], j \neq i$.

5.2 Analysis

5.2.1 Efficiency

We measure the computational complexity (running times) of our algorithms in group operations. Firstly, if we have CRS as 'powers-of-tau' $[\tau^i]_{1,2}$, it takes $O(M \log M)$ time to compute the 'powers-of-Lagrange' $[L_i(\tau)]_{1,2}$ via DFT. KGen takes $O(1)$ time. HintGen takes $O(M)$ multiplication and additions, respectively. Preprocess for $\mathcal{U}$ is dominated by $O(|\mathcal{U}|^2)$ additions. Enc and PartDec require $O(1)$ multiplications, while PartVerify requires $O(1)$ pairings. Finally, DecAggr require $O(|\mathcal{U}|\mathsf{polylog}|\mathcal{U}|)$ field operations and $O(|\mathcal{U}|)$ group multiplications (see Section 7 last paragraph).
For the communication complexity (sizes), we measure in group elements:

- $|\mathsf{CRS}| = (M+1)|\mathbb{G}_1| + (M+1)|\mathbb{G}_2|$,
- $|\mathsf{pk}_i| = 1|\mathbb{G}_1|$, $|\mathsf{sk}_i| = 1|\mathbb{Z}_p|$,
- $|\mathsf{hint}_i| = (M+3)|\mathbb{G}_1|$,
- $|\mathsf{ak}| = (5|\mathcal{U}| + 5)|\mathbb{G}_1|$, $|\mathsf{ek}| = 1|\mathbb{G}_1| + 1|\mathbb{G}_2|$,
- $|\mathsf{ct}| = 2|\mathbb{G}_1| + 7|\mathbb{G}_2| + 1|\mathbb{G}_T|$,
- $|\sigma_i| = 1|\mathbb{G}_2|$.

5.2.2 Correctness

For correctness, we first show that if the partial decryptions and the malicious keys sent by the adversary pass the verification, then they are well-formed.

- Malicious σ_i: If $\mathsf{PartVerify}(\mathsf{ct}, \sigma_i, \mathsf{pk}_i) = 1$ then by definition $\mathsf{pk}_i \circ [\gamma]_2 = [1]_1 \circ \sigma_i$. Since $\mathbb{G}_1$ is cyclic (since it has prime order p), there always exist $\mathsf{sk}_i \in \mathbb{Z}_p$ such that $\mathsf{pk}_i = [\mathsf{sk}_i]_1$ and $y \in \mathbb{Z}_p$ such that $\sigma_i = [y]_2$. Therefore, from the pairing we get that $[y]_T = [\gamma \mathsf{sk}_i]_T$, which means that $y = \gamma \mathsf{sk}_i$, thus $\sigma_i = [\gamma \mathsf{sk}_i]_2$ is well-formed.
- Malicious $\mathsf{pk}_i, \mathsf{hint}_i$: If $\mathsf{isValid}(\mathsf{CRS}, \mathsf{hint}_i, \mathsf{pk}_i) = 1$ then using the same argument $\mathsf{pk}_i = [\mathsf{sk}_i]$ for some $\mathsf{sk}_i \in \mathbb{G}_1$. Then by applying the five pairing checks and repeating the same line of thought as above it is straightforward to get that $\mathsf{hint}_i = \mathsf{HintGen}(\mathsf{CRS}, \mathsf{sk}_i, i, M)$ is well-formed.

At this point, correctness comes by careful inspection, in essence using the correctness of the univariate sumcheck (Lemma 1).

5.2.3 Security

Here, we prove the semantic security of our construction in the generic group model (we recall it in 3.3). Essentially, we prove that the decision problem related to our construction is generically hard.
For our proof, we need to show that it is computationally hard for a generic adversary that learns the elements specified by our construction (e.g., $\{[\tau^i]\}_i$, $\{\mathsf{sk}_i \tau_j\}_{i,j}$, ct^*, etc.) to distinguish between $[s_5]_T$ and a random element $[r]_T$.
For this, we employ the generic group model and the Master Theorem (Theorem 1 in appendix 3.3). In order to apply the theorem, we need to show the side condition holds. For this, in turn, we work in the symbolic group model to argue that S_5 is symbolically independent of the lists of (symbolic) elements that arise in the security game of our construction (and after the completion). Then hardness readily follows from the Master Theorem 1.[20] In conclusion, the core of the proof is in showing this independence.
For ease of presentation, we denote the set of non-corrupted users in the target universe $\mathcal{U}^*$ as $\mathcal{H} := \mathcal{U}^* \setminus \mathsf{Cor}$. 0 is by construction in $\mathcal{H}$ (it is always in the universe and cannot be corrupted); for clarity, we will write $\mathcal{H}^* = \mathcal{H} \setminus \{0\}$ as the set of honest indices excluding the 'artificial' index 0.

Theorem 2. *Construction 1 is a semantically secure Silent Threshold Encryption scheme in the generic group model.*

Proof. $\mathbf{Game}_0$: This is the semantic security game of Definition 5. The challenger $\mathcal{C}$ samples $\tau \leftarrow\!\$\ \mathbb{Z}_p^*$ computes $\mathsf{CRS} = ([\tau^1]_1, \ldots, [\tau^{M+1}]_1, [\tau^1]_2, \ldots, [\tau^{M+1}]_2)$ and gives the corresponding handles to $\mathcal{A}$, who answers with a target universe $\mathcal{U}^*$. Then $\mathcal{C}$ for each $i \in \mathcal{H}^*$ samples $\mathsf{sk}_i \leftarrow\!\$\ \mathbb{Z}_p^*$, computes $\mathsf{pk}_i = [\mathsf{sk}_i]_1, \mathsf{hint}_i \leftarrow \mathsf{HintGen}(\mathsf{CRS}, \mathsf{sk}_i, M, i)$ and sends the corresponding handles to $\mathcal{A}$. The adversary responds with $\{\mathsf{sk}_i\}_{i \in \mathsf{Cor}}$ and the challenger sends back the corresponding handles to $\{\mathsf{pk}_i, \mathsf{hint}_i\}_{i \in \mathsf{Cor}}$. The challenger runs $(\mathsf{ek}, \mathsf{ak}) \leftarrow \mathsf{Preprocess}(\mathsf{CRS}, \mathcal{U}^*, \{\mathsf{hint}_i, \mathsf{pk}_i\}_{i \in \mathcal{U}^*})$ and passes the handles of ek, ak to $\mathcal{A}$. $\mathcal{A}$ sends $\mathsf{msg}_0, \mathsf{msg}_1$ and t to $\mathcal{C}$. Then $\mathcal{C}$ samples a bit $b \leftarrow\!\$\ \{0,1\}$, generates the ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{msg}_b, t)$, i.e. samples $\gamma, s_1, \ldots, s_5 \leftarrow\!\$\ \mathbb{Z}_p^*$, computes the corresponding matrix $\boldsymbol{A}$ and sets $\mathsf{ct}^* = ([\gamma]_2, \boldsymbol{s}^\top \boldsymbol{A}, [s_5]_T + \mathsf{msg}_b)$, and sends the corresponding handles to $\mathcal{A}$. At this point the view of the adversary is $(M, t, \mathcal{U}^*, \{\mathsf{sk}_i\}_{i \in \mathsf{Cor}})$ and handles to:

$$(\mathsf{CRS}, \{\mathsf{pk}_i, \mathsf{hint}_i\}_{i \in \mathcal{H}}, \{\mathsf{pk}_i, \mathsf{hint}_i\}_{i \in \mathsf{Cor}}, (\mathsf{ek}, \mathsf{ak}), \mathsf{ct}^*)$$

Finally, $\mathcal{A}$ makes any generic operation of her choice to $\mathcal{C}$, who in turn answers them.

[20] Notice that for every PPT $\mathcal{A}$ $q = \mathsf{poly}(\lambda)$, $\nu = \mathsf{poly}(\lambda)$, $d = \mathsf{poly}(\lambda)$, while $p = \Theta(2^\lambda)$ in our groups.

$\mathbf{Game}_1$: Is the same as $\mathbf{Game}_0$ but we simplify the view of the adversary. First, instead of $\{\mathsf{hint}_i\}_i$, with the 'Lagrange polynomial'-hints, the challenger passes the 'powers-of-tau'-hints, $\{[\mathsf{sk}_i\tau]_1, \ldots, [\mathsf{sk}_i\tau^M]_1\}_i$, to the adversary. Second, $\mathcal{C}$ does not give $(\mathsf{ek}, \mathsf{ak})$ to $\mathcal{A}$, since this is deterministically computed, therefore, can be computed by $\mathcal{A}$ herself at the generic operations-query phase. Clearly, the adversary can compute the 'Lagrange-polynomial'-hints from the 'powers-of-tau'-hints and vice-versa; thus, $\Pr[\mathbf{Game}_0 = 1] = \Pr[\mathbf{Game}_1 = 1]$.
$\mathbf{Game}_2$: Here, we change the ciphertext to: $\mathsf{ct}^* = \left([\gamma]_2, \boldsymbol{s}^\top \boldsymbol{A}, [r]_T + \mathsf{msg}_b\right)$ where $r \leftarrow\!\$\ \mathbb{Z}_p^*$ is sampled uniformly at random by the challenger $\mathcal{C}$.
Now, it is clear that ct^* information theoretically leaks no information about the message msg_b; thus, $\Pr[\mathbf{Game}_2 = 1] = 1/2$.
Therefore, overall, the proof boils down to proving indistinguishability between $\mathbf{Game}_1$ and $\mathbf{Game}_2$. For this, we make use of the Master Theorem (Theorem 1). In order to apply it, we need to make sure that the 'side condition' holds. That is, our polymomial f is independent of our lists of polynomials $\boldsymbol{L}_1, \boldsymbol{L}_2, \boldsymbol{L}_T$. From now on, we will focus on proving this condition. Furthermore, we will work with the symbolic group representation, where group elements are represented as polynomials. Once we prove the independence condition in the symbolic experiment, we make use of the Master Theorem 1 to argue the indistinguishability of the actual games $\mathbf{Game}_1$ and $\mathbf{Game}_2$.
In our case, the formal variables that we concisely denote as $\boldsymbol{Y}$, are $\boldsymbol{Y} = (X, (K_i)_{i\in\mathcal{H}^*}, \Gamma, (S_i)_{i=1}^5)$ corresponding to the hidden–from the adversary–elements τ, $\{\mathsf{sk}_i\}_{i\in\mathcal{H}^*}$, γ, $\{s_i\}_{i=1}^5$ respectively. Concisely, we denote $\boldsymbol{K} = (K_i)_{i\in\mathcal{H}^*}$ and $\boldsymbol{S} = (S_i)_{i=1}^5$, thus $\boldsymbol{Y} = (X, \boldsymbol{K}, \Gamma, \boldsymbol{S})$. The lists of polynomials correspond to the view of the adversary, i.e. the handles she receives. The polynomial f corresponds to the element we want to prove pseudorandomness for. Overall:

$$\boldsymbol{L}_1(\boldsymbol{Y}) = \left(1, X, \ldots, X^{M+1}, \{K_i, K_iX, \ldots, K_iX^M\}_{i\in\mathcal{H}^*}, \overbrace{S_3}^{\boldsymbol{S}^\top\boldsymbol{a}_6}, \overbrace{\sum_{i\in\mathcal{H}^*} K_iL_i(X) + \sum_{i\in\mathsf{Cor}} \mathsf{sk}_iL_i(X) + S_4X^t + S_5}^{\boldsymbol{S}^\top\boldsymbol{a}_1}\right)$$

$$\boldsymbol{L}_2(\boldsymbol{Y}) = \left(1, X, \ldots, X^{M+1}, \Gamma, \overbrace{S_1 + \Gamma S_3}^{\boldsymbol{S}^\top\boldsymbol{a}_2}, \overbrace{S_1X^{M+1} - S_1}^{\boldsymbol{S}^\top\boldsymbol{a}_3}, \overbrace{S_1X + S_2X}^{\boldsymbol{S}^\top\boldsymbol{a}_4}, \overbrace{S_2}^{\boldsymbol{S}^\top\boldsymbol{a}_5}, \overbrace{S_4}^{\boldsymbol{S}^\top\boldsymbol{a}_7}, \overbrace{S_5X - S_5}^{\boldsymbol{S}^\top\boldsymbol{a}_8}\right)$$

$$\boldsymbol{L}_T(\boldsymbol{Y}) = (1)$$
$$f(\boldsymbol{Y}) = S_5$$

Concisely, we write $\boldsymbol{S}^\top\boldsymbol{a}_6$, $\boldsymbol{S}^\top\boldsymbol{a}_1$ for the two polynomials of $\boldsymbol{L}_1$ involving $\boldsymbol{S}$ (which are the ones that correspond to ct^*) and similarly $\boldsymbol{S}^\top\boldsymbol{a}_2$, $\boldsymbol{S}^\top\boldsymbol{a}_3$, $\boldsymbol{S}^\top\boldsymbol{a}_4$, $\boldsymbol{S}^\top\boldsymbol{a}_5$, $\boldsymbol{S}^\top\boldsymbol{a}_7$, $\boldsymbol{S}^\top\boldsymbol{a}_8$ the ones in $\boldsymbol{L}_2$.
First, we show that if f is dependent on $\boldsymbol{L} = (\boldsymbol{L}_1, \boldsymbol{L}_2, \boldsymbol{L}_T)$ then it should be the case that $\boldsymbol{S}^\top\boldsymbol{A}\cdot\boldsymbol{g}(X, \boldsymbol{K}) = \boldsymbol{S}^\top\boldsymbol{b}$, for a vector of polynomials $\boldsymbol{g} \in (\mathbb{Z}_p[X, (K_i)_{i\in\mathcal{H}^*}])^8$ that does not depend on $\boldsymbol{S}$. $f(\boldsymbol{Y}) = S_5$, so according to definition 2 if it is dependent on $\boldsymbol{L}$ then there exist $\kappa_i \in \mathbb{Z}_p$ such that: $S_5 = \sum_{i=1}^D \kappa_i g_i(\boldsymbol{Y})$, where $\boldsymbol{C}(\boldsymbol{L}) = (g_1(\boldsymbol{Y}), \ldots, g_D(\boldsymbol{Y}))$ is the completion of $\boldsymbol{L}$. The completion of $\boldsymbol{L}$ is essentially each term of $\boldsymbol{L}_1$ multiplied with each term of $\boldsymbol{L}_2$ (the tensor product). By inspection, we see that there are three types of terms in the completion:

- Terms that do not involve $\boldsymbol{S}$. E.g. $(K_iX^2)\cdot X$.
- Terms that involve $\boldsymbol{S}$ in both sides. E.g. $S_3\cdot(S_1 + \Gamma S_3)$.
- Terms that involve $\boldsymbol{S}$ only in one side. E.g. $S_3\cdot X^{M+1}$ or $K_iX\cdot(S_5X - S_5)$.

The first two types of elements cannot symbolically play any role in the sum $S_5 = \sum_{i=1}^D \kappa_i g_i(\boldsymbol{Y})$. The first ones have all degree 0 in $S_1, \ldots, S_5$ while the second ones all have quadratic terms $S_1, \ldots, S_5$. Thus we are only left with the third type of terms $S_5 = \sum_{i=1}^8 \boldsymbol{S}^\top\boldsymbol{a}_i g_i(X, \boldsymbol{K}, \Gamma)$, which can be written as: $\boldsymbol{S}^\top\boldsymbol{A}\cdot\boldsymbol{g}(X, (K_i)_{i\in\mathcal{H}^*}, \Gamma) = \boldsymbol{S}^\top\boldsymbol{b}$

Furthermore, if $\boldsymbol{S}^\top \boldsymbol{A} \cdot \boldsymbol{g}(X, (K_i)_{i\in\mathcal{H}^*}, \Gamma) = \boldsymbol{S}^\top \boldsymbol{b}$ then in fact it symbolically holds that $\boldsymbol{A} \cdot \boldsymbol{g}(X, (K_i)_{i\in\mathcal{H}^*}) = \boldsymbol{b}$.
Finally, we show that as long as $|\mathsf{Cor}| < t$ symbolically the above cannot happen. We prove this in the following lemma.

Lemma 2. *Assume that*

- $\boldsymbol{L}_1 = \left(1, \{X^j\}_{j\in[M+1]}, \{K_i X^j\}_{i\in\mathcal{H}}^{j\in[0,M]}\right)$,
- $\boldsymbol{L}_2 = \left(1, \{X^j\}_{j\in[M+1]}, \Gamma\right)$.

Let the matrix $\boldsymbol{A} \in \mathbb{Z}_p[X, (K_i)_{i\in\mathcal{H}^*}, \Gamma]^{5\times 8}$ *defined as:*

$$\boldsymbol{A} = \begin{pmatrix} F_1(\boldsymbol{Y}) & 1 & F_2(\boldsymbol{Y}) & X & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & X & 1 & 0 & 0 & 0 \\ 0 & \Gamma & 0 & 0 & 0 & 1 & 0 & 0 \\ X^t & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & F_3(\boldsymbol{Y}) \end{pmatrix}$$

(columns $1, 6$ *are in* $\boldsymbol{L}_1$ *and columns* $2, 3, 4, 5, 7, 8$, *are in* $\boldsymbol{L}_2$*), where* $F_1(\boldsymbol{Y}) = L_0(X) + \sum_{i\in\mathcal{H}^*} K_i L_i(X) + \sum_{i\in\mathsf{Cor}} \mathsf{sk}_i L_i(X)$, $F_2(\boldsymbol{Y}) = X^{M+1} - 1$, $F_3(\boldsymbol{Y}) = X - 1$ *and* $\boldsymbol{b} = (0, 0, 0, 0, 1)^\top$.
If $|\mathcal{H}^*| > M - t$ *then there does not symbolically exist any non-zero* $\boldsymbol{g} \in \mathbb{Z}_p[X, (K_i)_{i\in\mathcal{H}}, \Gamma]^8$ *(*g_1, g_6 *from* $\boldsymbol{L}_2$ *and* g_2, g_3, g_4, g_5, g_6, g_8 *from* $\boldsymbol{L}_1$ *resp.) such that* $\boldsymbol{A} \cdot \boldsymbol{g} = \boldsymbol{b}$.

Proof. Assume that there is a $\boldsymbol{g} = (g_i)_{i=1}^8$ such that $\boldsymbol{A} \cdot \boldsymbol{g} = \boldsymbol{b}$, then this translates to:

$$F_1(\boldsymbol{Z})g_1 + g_2 + g_3(X^{M+1} - 1) + g_4 X = 0 \tag{3}$$
$$g_4 X + g_5 = 0 \tag{4}$$
$$g_2\Gamma + g_6 = 0 \tag{5}$$
$$g_1 X^t + g_7 = 0 \tag{6}$$
$$g_1 + g_8(X - 1) = 1. \tag{7}$$

g_k for $i = 2, 3, 4, 5, 7, 8$ can be written as

$$g_k = \sum_{j=0}^{M+1} \alpha_j^{(k)} X^j + \sum_{i\in\mathcal{H}^*} \sum_{j=0}^{M} \beta_{i,j}^{(k)} K_i X^j,$$

while g_k for $i = 1, 6$ can be written as

$$g_k = \sum_{j=0}^{M+1} \alpha_j^{(k)} X^j + \delta^{(k)}\Gamma.$$

From construction the set $\mathcal{H}$ contains 0 with $\mathsf{sk}_0 = 1$; with $\mathcal{H}^*$ we denote the set $\mathcal{H} \setminus \{0\}$. Recall that $F_1(\boldsymbol{Z}) = \sum_{i\in\mathcal{H}^*} K_i L_i(X) + \sum_{i\in\mathsf{Cor}} \mathsf{sk}_i L_i(X) + L_0(X)$.
We further specify the structure of g_i's by inspecting Equations 3-7.

g_1**:** For g_1 we argue the following:

- Equation 3: Γ does not appear in equation 3 which means that $\delta^{(1)} = 0$. So

$$g_1 = g_1(X) = \sum_{j=0}^{M+1} \alpha_j^{(1)} X^j.$$

- Equation 6: We get that

$$\sum_{j=0}^{M+1} \alpha_j^{(1)} X^j \cdot X^t = \sum_{j=0}^{M+1} (-\alpha_j^{(7)}) X^j + \sum_{i \in \mathcal{H}^*} \sum_{j=0}^{M} (-\beta_{i,j}^{(7)}) K_i X^j$$

  so $\beta_{i,j}^{(7)} = 0$ for all i, j and $\alpha_j^{(1)} = 0$ for $j > M+1-t$, i.e. $g_1(X)$ has degree (at most) $M+1-t$, $g_1 = \sum_{j=0}^{M+1-t} \alpha_j^{(1)} X^j$.
- Equation 7: We get that

$$\sum_{j=0}^{M+1-t} \alpha_j^{(1)} X^j - 1 = \Big(\sum_{j=0}^{M+1} (-\alpha_j^{(8)}) X^j + \sum_{i \in \mathcal{H}^*} \sum_{j=0}^{M} (-\beta_{i,j}^{(8)}) K_i X^j \Big) (X-1)$$

  so $\beta_{i,j}^{(8)} = 0$ for all i, j and $g_8 = g_8(X)$. Therefore, $g_1(X) - 1 = g_8(X)(X-1)$ which means that $X - 1$ divides $g_1(X) - 1$, which in turns gives us that $g_1(1) := g_1(\omega^0) = 1$.

g_2: From Equation 5 we get that

$$\left(\sum_{j=0}^{M+1} \alpha_j^{(2)} X^j + \sum_{i \in \mathcal{H}^*} \sum_{j=0}^{M} \beta_{i,j}^{(2)} K_i X^j \right) \Gamma = \sum_{j=0}^{M+1} (-\alpha_j^{(6)}) X^j + (-\delta^{(6)}) \Gamma$$

which gives us that all $\alpha_j^{(2)}$, $\beta_{i,j}^{(2)}$, $\alpha_j^{(6)}$ are 0 except for $\alpha_0^{(2)}$. Therefore $g_6 = \delta^{(6)}\Gamma$ and $g_2 = \alpha_0^{(2)}$ is a constant.

g_4: From Equation 4 we get that

$$\left(\sum_{j=0}^{M+1} \alpha_j^{(4)} X^j + \sum_{i \in \mathcal{H}^*} \sum_{j=0}^{M} \beta_{i,j}^{(4)} K_i X^j \right) X = \left(\sum_{j=0}^{M+1} (-\alpha_j^{(5)}) X^j + \sum_{i \in \mathcal{H}^*} \sum_{j=0}^{M} (-\beta_{i,j}^{(5)}) K_i X^j \right)$$

thus $\alpha_0^{(5)} = 0$ and $\beta_{i,0}^{(5)} = 0$ for all i.

Now we use the Lemma 1 in Equation 3. We write the polynomial $g_1(X)$ in the lagrange basis form, $g_1(X) = \sum_{i=0}^{M+1} b_i L_i(X)$, where b_i are the evaluation in ω^i. As shown above $b_0 = g_1(\omega^0) = 1$, therefore

$$\sum_{i \in \mathcal{H}^*} K_i b_i + \sum_{i \in \mathsf{Cor}} \mathsf{sk}_i b_i + 1 + Q_x(X)X + Q_Z(X)(X^{M+1} - 1) = g_2 + g_3(X^{M+1} - 1) + g_4 X :=$$

$$:= \alpha^{(2)} + \Big(\sum_{j=0}^{M+1} \alpha_j^{(3)} X^j + \sum_{i \in \mathcal{H}^*} \sum_{j=0}^{M} \beta_{i,j}^{(3)} K_i X^j \Big) (X^{M+1} - 1) + \sum_{j=1}^{M+1} (-\alpha_j^{(5)}) X^j + \sum_{i \in \mathcal{H}^*} \sum_{j=1}^{M} (-\beta_{i,j}^{(5)}) K_i X^j$$

where above we replaced g_2 and $g_4 X$. By inspection we observe that $\beta_{i,j}^{(3)} = 0$ for all i, j, since $\beta_{i,j}^{(3)} K_i X^{j+M}$ does not appear anywhere in the equation. Furthermore, $\beta_{i,j}^{(5)} = 0$ for all i, j, so:

$$b_i = 0, \quad \text{for each } i \in \mathcal{H}^*.$$

This, in addition to $b_0 = 1$ shown before, gives us that the degree of $g_1(X) = \sum_{i=0}^{M+1} b_i L_i(X)$ is $\deg(g_1) \geq |\mathcal{H}^*| + 1$. On the other hand, we also argued above that $M + 1 - t \geq \deg(g_1)$, thus $|\mathcal{H}^*| + 1 \leqslant M + 1 - t$. Now, by assumption $|\mathcal{H}^*| > M - t$, which is a contradiction. □

Therefore, we conclude that f is independent of $\boldsymbol{L}_1, \boldsymbol{L}_2, \boldsymbol{L}_T$. So, we can apply the Master Theorem 1 that gives us:

$$\left| \Pr[\mathbf{Game}_1 = 1] - \Pr[\mathbf{Game}_2 = 1] \right| \leqslant \frac{q + (|\mathcal{H}^*| + 2)(M+1) + 10}{2p} = \mathsf{negl}(\lambda)$$

which, since $\Pr[\mathbf{Game}_2 = 1] = 1/2$ gives that $\Pr[\mathbf{Game}_0] = \frac{1}{2} + \mathsf{negl}(\lambda)$ and concludes the proof. □

6 Extensions of the Basic Construction

We present various extensions and implications of our core silent threshold encryption scheme described in Section 5.

6.1 Multiverse Silent Threshold Encryption

As described in the introduction, a highly desirable property for a threshold encryption scheme is to be able to support different sets of decryptor committees, i.e., different 'universes', without re-running the setup of the system. This is reminiscent of multiverse threshold signtures [BGJ+23, GJM+24].
We stress that our construction supports *by default* any universe. We specify at the beginning an upper bound on the size of the universe M and then the scheme works for any subset of parties of size $\leqslant M$. Each user is computing her hint, which is then valid for *every* universe. Then, notice that the Preprocess algorithm takes as input any universe. Crucially, the efficiency of our scheme only scales with the maximum number of universe M, but not the overall number of users participating in the system. For instance, even if we have a million users, if the maximum universe size is 1024, each user only needs to publish hints[21] of size $\leqslant 1024$.

6.2 CCA2 Security

In CCA2 security for threshold encryption [BBH06], the adversary $\mathcal{A}$ is given access to the partial decryption oracle for honest users before and after she receives the challenge ciphertext ct^*. Intuitively, to enhance an STE with CCA2 security, we need to make sure that ct^* is not *indirectly* queried for partial decryption after $\mathcal{A}$ sees it. Of course, for the security definition to be meaningful the adversary cannot query *directly* ct^*.
A bit more formally, the ciphertext $\mathsf{ct} = ([\gamma]_2, \mathbf{ct}_2, \mathsf{ct}_3)$. The partial decryption, nonetheless, is oblivious to $\mathbf{ct}_2, \mathsf{ct}_3$, it solely depends on $[\gamma]_2$. What an adversary can possibly do after receiving the challenge ciphertext $\mathsf{ct}^* = ([\gamma^*]_2, \mathbf{ct}_2^*, \mathsf{ct}_3^*)$ is to query $\mathsf{ct}' = ([\gamma^*]_2, \mathbf{ct}_2', \mathsf{ct}_3')$, which is technically a different ciphertext.
To prevent this and achieve CCA2 security, it suffices to 'bind' together the three parts of the ciphertext. For this, we employ a straight-line simulation-extractable [Sah99, DDO+01] tag-based NIZK as was done in [SG02] for the ElGamal encryption scheme. The NIZK proves knowledge of γ, while the tag contains $\mathbf{ct}_2$ and ct_3. This can be, for instance, a straight-line simulation-extractable Schnorr protocol for $[\gamma]_2$ in the Algebraic Group Model, where we use Fiat-Shamir, and the random oracle additionally takes the $\mathsf{tag} = (\mathbf{ct}_2, \mathsf{ct}_3)$ as input. Then, the ciphertext of the CCA2 secure scheme gets $([\gamma]_2, \mathbf{ct}_2, \mathsf{ct}_3, \pi)$, where π is the NIZK proof. After this appropriate adaptation, it is straightforward for the challenger to simulate partial decryption-oracle queries: She uses the extractor of the NIZK to obtain $[\gamma]_2$ and then computes the partial decryption $\mathsf{sk}_i \cdot [\gamma]_2$ honestly. The only difference between CPA and CCA security game is the adversary's partial decryption query, once we can simulate these queries, the rest of the proof is similar to CPA security. Since this transformation is a standard technique, we only describe it at a high level.

[21] The hints can be made independent of the index i as described in Section 6.3.

6.3 Flexible Broadcast (and Threshold) Encryption

Freitag et al. [FWW23] recently introduced the notion of *Flexible Broadcast Encryption* (FBE). This is, essentially, an enhancement of Distributed Broadcast Encryption (DBE) [WQZD10, BZ14], i.e. Brodacast Encryption without a central authority (see Section 1.2), in which the user does not need to be assigned to a specific index in $i \in [M]$. That is, the public keys of the users are oblivious to any index i. Then, the public keys alone suffice to encrypt and decrypt. [FWW23] provided a construction from general-purpose witness encryption. Then Garg et al. [GLWW23] provided a generic compiler for FBE from any DBE, however inducing an $\omega(\log \lambda)$ overhead on the size of the public keys. Here we present a direct FBE construction from Pairings with $O(1)$ keys.

For ease of presentation we recall the Flexible Broadcast Encryption definition from [FWW23] (with adapted notation to fit ours) in Appendix A.

With a simple modification, our construction can achieve this property, where the public key and the corresponding hints do not depend on any index i. First, we observe that the actual public key already does not depend on the index of the user: $\mathsf{pk} = [\mathsf{sk}]_1$. The hint hint_i, however, depend on the specific index: for instance $[\mathsf{sk}_i L_i(\tau)]_1$ depends on the specific lagrange poynomial L_i. To circumvent this we observe that a 'powers-of-tau' hint $\mathsf{hint}'_i = \{[\mathsf{sk}_i \tau^j]_1\}_{j\in[M+1]}$ is functionally fully equivalent; from hint'_i one can efficiently derive hint_i. Therefore, the modified STE scheme with hint'_i as hints are equivalent, in terms of correctness, with Construction 1. We note that the observation, regarding hint'_i being i-independent and equivalent to hint_i, was already made in [GJM+24] in the context of Silent Threshold Signatures.

6.3.1 Our Construction

For completeness we show our FBE construction below.

- $\mathsf{Setup}(1^\lambda)$: Sample $\tau \leftarrow\!\!\$\ \mathbb{Z}_p$ and output:
$$\mathsf{CRS} = \left([\tau^1]_1, \ldots, [\tau^{M+1}]_1, [\tau^1]_2, \ldots, [\tau^{M+1}]_2\right).$$
- $\mathsf{KGen}(1^\lambda)$: Sample $x \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and output $\mathsf{pk} = [x]_1$, $\mathsf{sk} = x$.
- $\mathsf{HintGen}(\mathsf{CRS}, \mathsf{sk}, M)$: output
$$\mathsf{hint}_{\mathsf{pk}} = \left([\mathsf{sk}\cdot\tau]_1, \ldots, [\mathsf{sk}\cdot\tau^{M+1}]_1\right)$$
- $\mathsf{Preprocess}(\mathsf{CRS}, \{\mathsf{hint}_{\mathsf{pk}_i}, \mathsf{pk}_i\}_{i=1}^k)$: Verify the validity of each hint: for each $i \in [k]$ run $\mathsf{isValid}(\mathsf{CRS}, \mathsf{hint}_{\mathsf{pk}_i}, \mathsf{pk}_i)$ ($\mathsf{isValid}$ is defined below) and let $V \subseteq [k]$ be the set of the indices with valid hints. Set $\mathsf{sk}_0 = 1$ and $\mathsf{sk}_i = 0$ for each $i \notin V$. Output:
$$\mathsf{dk} = \left(V, \left\{\mathsf{pk}_i\right\}_{i\in V\cup\{0\}}, \left\{[\mathsf{sk}_i\big(L_i(\tau) - L_i(0)\big)]_1\right\}_{i\in V\cup\{0\}}, \left\{\left[\mathsf{sk}_i \frac{L_i^2(\tau) - L_i(\tau)}{Z(\tau)}\right]_1\right\}_{i\in V\cup\{0\}},\right.$$
$$\left.\left\{\left[\mathsf{sk}_i \frac{L_i(\tau) - L_i(0)}{\tau}\right]_1\right\}_{i\in V\cup\{0\}}, \left\{\left[\sum_{j\in S, j\neq i} \mathsf{sk}_j \frac{L_i(\tau)L_j(\tau)}{Z(\tau)}\right]_1\right\}_{i\in V\cup\{0\}}\right).$$
$$\mathsf{ek} = \left(\left[\sum_{i\in V} \mathsf{sk}_i L_i(\tau)\right]_1, [Z(\tau)]_2\right) := (C, Z)$$

  It is crucial that the above can be computed by having access to $\left([\mathsf{sk}_i\tau^j]_1\right)_{j\in[k+1]}$ and $[\mathsf{sk}_i]_1$ (which is contained in $\mathsf{hint}_{\mathsf{pk}} = \left([\mathsf{sk}_i\tau^j]_1\right)_{j\in[M+1]}$ and pk_i respectively). In essence, from $\left([\mathsf{sk}_i\tau^j]_1\right)_{j\in[0,k+1]}$ one can efficiently compute any $[\mathsf{sk}\cdot f(\tau)]_1$ for *any* univariate polynomial f of degree at most $k+1$

- $\mathsf{Enc}(\mathsf{ek}, \mathsf{msg})$: Identical to Construction 1 with threshold set to 1.
- $\mathsf{Dec}(\mathsf{dk}, \mathsf{ct}, (j, \mathsf{sk}_j))$: Each user runs $\sigma_j \leftarrow \mathsf{PartDec}(\mathsf{sk}, \mathsf{ct})$ and outputs $\mathsf{msg}^* \leftarrow \mathsf{DecAggr}(\mathsf{CRS}, \mathsf{dk}, \mathsf{ct}, \sigma_j)$, where DecAggr is defined in Construction 1.

To conclude our FBE construction we define the isValid algorithm as follows:

$\mathsf{isValid}(\mathsf{CRS}, \mathsf{hint}_{\mathsf{pk}}, \mathsf{pk}) \rightarrow \{0,1\}$: parses $\mathsf{hint}_{\mathsf{pk}} := (h_1, \ldots, h_{M+1})$ and output 1 iff it holds that:

1. $h_1 \circ [1]_2 = \mathsf{pk} \circ [\tau]_2$
2. $h_i \circ [1]_2 = h_{i-1} \circ [\tau]_2$, for each $i \in [2, M+1]$

6.3.2 Security

For semantic security, observe that the security proof already uses a modified scheme with 'powers-of-tau' hints in a game-hop, it is essentially $\mathbf{Game}_1$. Therefore, the security of this modification readily follows from that.

Remark 4. *We describe our construction in the context of FBE, to stress the implications of our STE scheme to this recently introduced notion. That is we describe it for a threshold of $T = 1$. We note that our construction can be extended in a straightforward way to work for any threshold T. Overall, this yields the first pairing-based Flexible Threshold Encryption scheme.*

6.4 Forward and Post-Compromise Security

As a final extension, in this subsection, we briefly discuss how we can achieve Threshold Encryption with Silent Setup that additionally supports Forward Security [BM99] and Post-Compromise Security [OY91]. These are two highly desirable properties in real-world applications where secret keys can be potentially compromised.

Forward Secure STE. A Threshold Encryption scheme has, informally, Forward Security if a secret key compromise at the present cannot affect old ciphertexts. That is, a leakage of a secret key should not allow one to partially decrypt (sufficiently) old ciphertexts. This, intuitively, requires that users' keys are periodically updated in manner that the new keys cannot decrypt ciphertexts computed with the old keys. In this section, we informally discuss how we can obtain a Forward Secure STE scheme from Pairings using the techniques we developed for our STE scheme of section 5.
Following our reasoning for STE, if we start from a Pairing-based Threshold Signature scheme that has (1) silent setup, (2) public linear verification (in the pairing operator, '$\circ$'), and additionally has (3) *Forward Security*, we can use our witness encryption methodology to construct an STE with Forward Security.
To this end, we can resort to Pixel [DGNW20], a Forward Secure aggregate signature scheme, which is a variant of BLS. The verification equation of an aggregated signature in Pixel is similar to the one of BLS:

$$[\alpha]_2 \circ \mathsf{aPK} + [\alpha_0 + \sum_{j=1}^{\mathbf{t}} \alpha_j t_j + \alpha_{\ell+1}\gamma]_1 \circ \sigma_2^* = \sigma_1^* \circ [1]_2$$

where $\sigma^* = (\sigma_1^*, \sigma_2^*) = (\prod_{i=1}^n \sigma_{i,1}, \prod_{i=1}^n \sigma_{i,2})$ is the aggregated signature of $\sigma_1, \ldots, \sigma_n$, $\{[\alpha]_2, [\alpha_0]_1, \ldots, [\alpha_{\ell+1}]_1\}$ are public group elements, $[\gamma]_1$ is the tag and $\mathbf{t}$ a public variable specifying the session number of the signature. Crucially the aggregated public key $\mathsf{aPK} = [x_1 + \ldots + x_n]_2$ is identical to the one in the BLS signature. Therefore, one can build a Silent Setup version of Pixel analogously to the STS scheme of BLS, unveiled in the technical overview (see sec. 2.2). The verification of the Silent Setup Pixel would be as follows:

1. $[\mathsf{SK}(\tau)]_1 \circ [B(\tau)]_2 = [1]_2 \circ \mathsf{aPK} + [Z(\tau)]_2 \circ [Q_Z(\tau)]_1 + [\tau]_2 \circ [Q_x(\tau)]_1$
2. $[\tau]_2 \circ [Q_x(\tau)]_1 = [1]_2 \circ [\hat{Q}_x(\tau)]_1$
3. $[\alpha]_2 \circ \mathsf{aPK} + [\alpha_0 + \sum_{j=1}^{\mathbf{t}} \alpha_j t_j + \alpha_{\ell+1}\gamma]_1 \circ \sigma_2^* = \sigma_1^* \circ [1]_2$
4. $[\tau^t]_1 \circ [B(\tau)]_2 = [1]_2 \circ [\hat{B}(\tau)]_1$
5. $[1]_1 \circ [B(\tau)]_2 = [\tau - 1]_2 \circ [Q_0(\tau)]_1 + 1$

Finally, we observe that the above is a public linear constraint system, thus from this STS scheme we can build a Threshold Encryption scheme with Silent Setup that admits Forward Security. The latter property is inheritted directly from the forward security of Pixel.

Post-Compromise Secure STE. Post-Compromise Security in the context of Threshold Encryption is, essentially, the property that even if a secret key is leaked in a specific time instance, it will become shortly useless in (partially) decrypting ciphertexts. In particular, to achieve this property users' keys should be periodically updated.
We stress that Silent Setup offers a straightforward means to Post-Compromise Security, in comparison to DKG protocols. That is, a party is updating her keys by just locally sampling a new secret and then (deterministically) computing the corresponding public keys and hints. After that, the user posts her updated public key/hint pair and new ciphertexts can be computed with respect to the updated keys of the user. In contrast, in Threshold Encryption schemes that resort to Distributed Key Generation protocols a new round of interaction between the users should happen in order to update the users' keys.

7 Implementation and Evaluation

To evaluate the concrete performance of our silent-threshold encryption scheme, we implement our scheme in Rust. We use the `arkworks` [ac22] library for implementations of pairing-friendly curves and the associated algebra. Our code can be found at https://github.com/guruvamsi-policharla/silent-threshold.

Setup. All of our experiments were run on a 2019 MacBook Pro with a 2.4 GHz Intel Core i9 processor and 16 GB of DDR4 RAM in single-threaded mode.[22] We use BLS12-381 as our pairing-friendly curve. For all experiments, we use a threshold $t = n/2$, where n is the number of parties.

Evaluation. We now evaluate the performance of our STE. In particular, we aim to answer the following questions and provide insights about bottlenecks in different parts of the protocol.

- How long does it take to set up and aggregate public keys? What are their sizes?
- How long does it take to encrypt a message? What is the size of corresponding ciphertexts?
- How long does it take to recover all messages given partial decryptions?

Finally, how do the above vary with committee size? Throughout the evaluation, where meaningful, we compare our scheme against the ElGamal encryption scheme to understand the *price* of silent setup.

Key Generation. The local key generation is only done once per party, and for a large committee of size 1024, it takes less than 28 s. This local key generation cost scales linearly with the number of parties.

Parties	Key Gen. (s)	Decryption (ms)
8	0.007	12.9
32	0.06	20.2
128	0.65	45.5
512	7.87	126.1
1024	27.79	211.0

Table 1: Scaling of key generation and reconstruction times with committee size.

Although we have improved performance over the original hinTS implementation [GJM+24], the code can be optimized further. We provide the local key generation times for smaller committee sizes in Table 1.

Encryption. Given an aggregated public key, a constant number of group operations are needed to encrypt a ciphertext. The ciphertext consists of 7 $\mathbb{G}_2$ elements and 2 $\mathbb{G}_1$ elements and a string proportional to the message length, which amounts to 768 bytes ignoring the message size. In comparison to ElGamal, which requires an expensive interactive setup, our ciphertexts are only $12\times$ larger. We observe that it takes < 7 ms to encrypt a message, independent of the committee size. There is a trade-off possible between the size of the public key and ciphertexts by switching the source groups. In this case, the ciphertext size would be 528 bytes, and the encryption would be faster.

Partial Decryption. Here, each party computes one group exponentiation which takes < 1 ms, and the size of each partial decryption is just one $\mathbb{G}_2$ element (96 bytes), which can again be halved by switching the groups. This makes the communication needed for decryption quite small – in fact, identical to ElGamal.

Reconstruction. Any party can publicly recover the message given enough partial decryptions of the ciphertext. The main bottleneck here is the group exponentiations involved in computing various witness elements $\boldsymbol{w}$, which scale linearly with the number of parties.[23] `arkworks` provides an implementation of Pippenger's algorithm [Pip80], which brings the complexity of multi-scalar-multiplications down to $O(n/\log n)$. Even for a committee of size 1024, ciphertexts can be decrypted in close to 200 ms. Indeed, ElGamal only needs one multi-scalar-multiplication of size $O(n)$ for reconstruction and will be more efficient than our scheme.
We note that, when interpolating the polynomial $B(X)$, we are only given evaluations over a subset of the roots of unity, which do not necessarily form a set FFT-friendly evaluation points. Interpolating this polynomial naïvely using Lagrange interpolation, would take $O(n^2)$ *field operations*. This is indeed what we do in our implementation. However, one can speed up this to $O(n\,\mathrm{polylog}(n))$ using standard techniques for non-FFT friendly evaluation points (see, e.g., [HHL17, Sch71]). We do not implement this faster version because, for the committee sizes (e.g., 1024) that we benchmark, $O(n^2)$ field operations are still much faster than $O(n)$ group operations. Actually, the interpolation cost is only $< 1\%$ of the total reconstruction time. As a result, it would not meaningfully affect the reported numbers.

Acknowledgments

The first, third, and fourth authors are supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, Visa Inc, the BAIR Commons Meta Fund, and the Stellar Development Foundation. The third author is also supported by the AI Policy Hub Fellowship from the UC Berkeley Center for Long-Term Cybersecurity.

[22] We note that our scheme can take can full advantage of multi-threaded architectures to achieve a perfect division of work.
[23] Since aggregation takes all partial descriptions as input, its cost inevitably scales with n.

The second author received funding from projects from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283) and from the Spanish Government under projects PRODIGY (TED2021-132464B-I00) and ESPADA (PID2022-142290OB-I00). The last two projects are co-funded by European Union EIE, and NextGenerationEU/PRTR funds.

References

[ac22] arkworks contributors. `arkworks` zksnark ecosystem. https://arkworks.rs, 2022. 28

[AJM+21] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 363–373, New York, NY, USA, 2021. Association for Computing Machinery. 3

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, Heidelberg, May 2005. 13

[BBH06] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 226–243. Springer, Heidelberg, February 2006. 25

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019. 14

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001. 8

[BFF+14] Gilles Barthe, Edvard Fagerholm, Dario Fiore, John C. Mitchell, Andre Scedrov, and Benedikt Schmidt. Automated analysis of cryptographic assumptions in generic group models. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 95–112. Springer, Heidelberg, August 2014. 14

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001. 3, 4

[BGJ+23] Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Threshold signatures in the multiverse. IEEE S&P 2023, 2023. https://eprint.iacr.org/2023/063. 3, 25

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001. 8, 13

[BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448. Springer, Heidelberg, August 1999. 27

[BO22] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Report 2022/898, 2022. https://eprint.iacr.org/2022/898. 6

[Boy08] Xavier Boyen. The uber-assumption family (invited talk). In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 39–56. Springer, Heidelberg, September 2008. 13

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499. Springer, Heidelberg, August 2014. 4, 5, 7, 26

[CES21] Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. In Maura B. Paterson, editor, *18th IMA International Conference on Cryptography and Coding*, volume 13129 of *LNCS*, pages 129–157. Springer, Heidelberg, December 2021. 7

[CJW22] Agostino Capponi, Ruizhe Jia, and Ye Wang. The evolution of blockchain: from lit to dark. *arXiv preprint*, 2022. 6

[CNR+22] Matteo Campanelli, Anca Nitulescu, Carla Ràfols, Alexandros Zacharakis, and Arantxa Zapico. Linear-map vector commitments and their practical applications. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 189–219. Springer, Heidelberg, December 2022. 14

[DCX+23] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bunz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. CCS 2023, 2023. https://eprint.iacr.org/2023/598. 3, 4, 7

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg, August 2001. 25

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Heidelberg, August 1988. 3

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, August 1990. 3

[DGK+20] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy*, pages 910–927. IEEE Computer Society Press, May 2020. 6

[DGNW20] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2093–2110. USENIX Association, August 2020. 27

[DHMR07] Vanesa Daza, Javier Herranz, Paz Morillo, and Carla Ràfols. CCA2-secure threshold broadcast encryption with shorter ciphertexts. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 35–50. Springer, Heidelberg, November 2007. 3, 6

[DHMR08] Vanesa Daza, Javier Herranz, Paz Morillo, and Carla Ràfols. Ad-hoc threshold broadcast encryption with shorter ciphertexts. *Electronic Notes in Theoretical Computer Science*, 192(2):3–15, 2008. 3, 6

[DHMW23] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: Verifiable encryption to the future made practical. Financial Crypto 2023, 2023. https://eprint.iacr.org/2022/433. 6, 8

[DKL+23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 417–446. Springer, Heidelberg, April 2023. 7

[DP08] Cécile Delerablée and David Pointcheval. Dynamic threshold public-key encryption. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 317–334. Springer, Heidelberg, August 2008. 3, 6

[DP23] Pratish Datta and Tapas Pal. Registration-based functional encryption. *Cryptology ePrint Archive*, 2023. 7

[DXR21] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2705–2721. ACM Press, November 2021. 3

[DYX+22] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2518–2534, 2022. 3

[eth] Ethereum: Minimal Light Client. https://github.com/ethereum/annotated-spec/blob/master/altair/sync-protocol.md. 7

[FFM+23] Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. *Cryptology ePrint Archive*, 2023. 7

[FKdP23] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo commitments: Registration-based encryption and key-value map commitments for large spaces. In *Asiacrypt 2023-International Conference on the Theory and Application of Cryptology and Information Security*, 2023. 7

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer, Heidelberg, August 1994. 5

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. 4

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered abe, flexible broadcast, and more. CRYPTO 2023, 2023. https://eprint.iacr.org/2023/812. 4, 5, 7, 26, 36

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. 4, 8

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 63–93. Springer, Heidelberg, April 2019. 7

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 689–718. Springer, Heidelberg, November 2018. 7

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20:51–83, 1999. 3

[GJM+24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hints: Threshold signatures with silent setup. IEEE S&P 2024, 2024. https://eprint.iacr.org/2023/567. 3, 4, 7, 9, 10, 12, 14, 18, 25, 26, 29

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1065–1079, 2023. 7

[GKW+16] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016. 6

[GLWW23] Rachit Garg, George Lu, Brent Waters, and David J Wu. Realizing flexible broadcast encryption: How to broadcast to a public-key directory. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1093–1107, 2023. 4, 5, 7, 26

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016. 7

[GV20] Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 621–651. Springer, Heidelberg, August 2020. 7

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953. 7

[HHL17] David Harvey, Joris Van Der Hoeven, and Grégoire Lecerf. Faster polynomial multiplication over finite fields. *Journal of the ACM (JACM)*, 63(6):1–23, 2017. 29

[HLR10] Javier Herranz, Fabien Laguillaumie, and Carla Ràfols. Constant size ciphertexts in threshold attribute-based encryption. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 19–34. Springer, Heidelberg, May 2010. 3, 6

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 511–542. Springer, Heidelberg, April 2023. 7

[JSSW21] Aljosha Judmayer, Nicholas Stifter, Philipp Schindler, and Edgar R. Weippl. Estimating (miner) extractable value is hard, let's go shopping! *IACR Cryptology ePrint Archive*, 2021. 6

[KKMS20] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. New York, NY, USA, 2020. Association for Computing Machinery. 3

[KLJD23] Alireza Kavousi, Duc V. Le, Philipp Jovanovic, and George Danezis. Blindperm: Efficient mev mitigation with an encrypted mempool and permutation. Cryptology ePrint Archive, Paper 2023/1061, 2023. https://eprint.iacr.org/2023/1061. 6

[KMS20] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1751–1767. ACM Press, November 2020. 3

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. Cryptology ePrint Archive, Paper 2023/874, 2023. https://eprint.iacr.org/2023/874. 4, 5, 7

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010. 4, 10

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005. 4, 12, 13

[MRV+21] Silvio Micali, Leonid Reyzin, Georgios Vlachos, Riad S. Wahby, and Nickolai Zeldovich. Compact certificates of collective knowledge. In *2021 IEEE Symposium on Security and Privacy*, pages 626–641. IEEE Computer Society Press, May 2021. 7

[MS22] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag, 2022. 6

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 51–59, 1991. 27

[PFW22] Julien Piet, Jaiden Fairoze, and Nicholas Weaver. Extracting godl [sic] from the salt mines: Ethereum miners extracting value, 2022. 6

[Pip80] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2):230–250, 1980. 29

[PNS23] Julien Piet, Vivek Nair, and Sanjay Subramanian. Mevade: An mev-resistant blockchain design. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2023. 6

[QZG21] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? *arXiv preprint*, 2021. 6

[RK23] Antoine Rondelet and Quintus Kilbourn. Threshold encrypted mempools: Limitations and considerations, 2023. 6

[RSY21] Leonid Reyzin, Adam D. Smith, and Sophia Yakoubov. Turning HATE into LOVE: compact homomorphic ad hoc threshold encryption for scalable MPC. In *Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8-9, 2021, Proceedings*, pages 361–378, 2021. 3, 4, 6, 7

[RZ21] Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Heidelberg. 14

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999. 25

[Sch71] Arnold Schonhage. Schnelle multiplikation grosser zahlen. *Computing*, 7:281–292, 1971. 29

[SG02] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, March 2002. 25

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. 4, 12, 13

[TC+21] Christof Ferreira Torres, Ramiro Camino, et al. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium*, 2021. 6

[TCZ+20] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 877–893, 2020. 3

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 535–559. Springer, Heidelberg, August 2022. 8

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 195–221. Springer, Heidelberg, December 2022. 8

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption (poster presentation). In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 741–743. ACM Press, October 2010. 4, 5, 7, 26

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered abe via predicate encodings. *Cryptology ePrint Archive*, 2023. 7

A Flexible Broadcast Encryption Definition

We recall the definition of Flexible Broadcast Encryption that was put forth by Freitag et al. [FWW23]. For consistency, we adapt the definition to our notation and syntax. In particular, we enrich the FBE syntax with two additional algorithms, HintGen and Preprocess, analogously to our STE definition. Our FBE definition implies the original one by [FWW23] if one integrates HintGen in KGen and Preprocess in Enc and Dec. We note that a preprocessing algorithm for a set of public keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_k\}$ is implicitly performed inside Enc and Dec in all previous FBE constructions.

Definition 6 (FBE)**.** *A Flexible Broadcast Encryption scheme consists of a tuple of algorithms* $\Sigma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{HintGen}, \mathsf{Preprocess}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

- $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, M)$*: On input the security parameter* λ *and an upper bound* M *on the maximum number of users, the* Setup *algorithm outputs a common reference string* CRS*.*
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$*: On input the security parameter* λ*, the* KGen *algorithm outputs a public/secret key pair* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{hint}_{\mathsf{pk}} \leftarrow \mathsf{HintGen}(\mathsf{CRS}, \mathsf{sk}, M)$*: On input the* CRS*, the secret key* sk*, the maximum number of parties* M*, the* HintGen *algorithm outputs a hint* $\mathsf{hint}_{\mathsf{pk}}$ *for the corresponding to* sk *public key.*
- $(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{Preprocess}(\mathsf{CRS}, \{\mathsf{hint}_{\mathsf{pk}_i}, \mathsf{pk}_i\}_{i=1}^k)$*: On input the* CRS*, a set of public key/hint pairs* $\{\mathsf{hint}_{\mathsf{pk}_i}, \mathsf{pk}_i\}_{i=1}^k$ *(where* $k \leqslant M$*), the* Preprocess *algorithm computes a decryption key* dk *and an encryption key* ek*.*
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{msg})$: *An encryption key* ek *and a message* msg*, it outputs a ciphertext* ct*.*
- $\mathsf{msg} \leftarrow \mathsf{Dec}(\mathsf{dk}, \mathsf{ct}, (j, \mathsf{sk}_j))$: *On input a decryption key* dk*, a ciphertext* ct *and an index and secret key pair* (j, sk_j) *(where* $j \in [k]$*), it outputs a message* msg*.*

Moreover, FBE must have succinct (on the number of public keys k*) ciphertexts.*

Remark 5. *The algorithm* HintGen *could be equivalently integrated in the* KGen *algorithm and* $\mathsf{hint}_{\mathsf{pk}}$ *integrated to* pk *correspondingly. Similarly* Preprocess *could be integrated to* Enc *and* Dec *so that* ek *is computed by* Enc *and* dk *by* Dec*.*

Analogously to [FWW23] we define static security for FBE as follows.[24]

Definition 7 (FBE Static Security)**.** *An FBE scheme* Σ *satisfies static security if, for every* $M = \mathsf{poly}(\lambda)$ *and any adversary PPT* $\mathcal{A}$*, the output of the game in Figure 2 is* 1 *with probability* $\leqslant 1/2 + \mathsf{negl}(\lambda)$*.*

[24] For ease of presentation we define only static security. Our scheme of section 6.3 can be also proven secure in the GGM under the stronger notion of semi-static security.

1. *The challenger runs* $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda, M)$ *and gives* CRS *to* $\mathcal{A}$.
2. *The adversary picks* k*, the size of the broadcast set of the challenge ciphertext.*
3. *The challenger samples* k *public keys and the corresponding hints,* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(1^\lambda)$ *and* $\mathsf{hint}_i = \mathsf{HintGen}(\mathsf{CRS}, \mathsf{sk}_i, M)$*, for* $i \in [k]$*, and sends them to* $\mathcal{A}$.
4. *For all* $i \in [k+1, M]$*, the adversary picks a public key* pk_i *and the corresponding hint* hint_i.
5. *The challenger invokes the preprocessing as* $(\mathsf{dk}, \mathsf{ak}) \leftarrow \mathsf{Preprocess}(\mathsf{CRS}, \{\mathsf{hint}_{\mathsf{pk}_i}, \mathsf{pk}_i\}_{i=1}^k)$ *and the output are given to* $\mathcal{A}$.
6. *The adversary picks messages* $\mathsf{msg}_0, \mathsf{msg}_1$.
7. *The challenger picks a bit* $b \leftarrow \{0,1\}$ *and generates a ciphertext* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{msg}_b)$.
8. *The adversary outputs a bit* b' *and wins the semantic security game if* $b' = b$*, in which case, the output of the game is 1.*

Figure 3: FBE Static Security Game