Realizing Flexible Broadcast Encryption: How to Broadcast to a Public-Key Directory

Rachit Garg[1], George Lu[1], Brent Waters[1,2], and David J. Wu[1]

[1]UT Austin
{rachg96, gclu, bwaters, dwu4}@cs.utexas.edu
[2]NTT Research

Abstract

Suppose a user wants to broadcast an encrypted message to K recipients. With public-key encryption, the sender would construct K different ciphertexts, one for each recipient. The size of the broadcasted message then scales linearly with K. A natural question is whether the sender can encrypt the message with a ciphertext whose size scales *sublinearly* with the number of recipients.

Broadcast encryption offers one solution to this problem, but at the cost of introducing a central *trusted* party who issues keys to different users (and correspondingly, has the ability to decrypt all ciphertexts). Recently, several works have introduced notions like distributed broadcast encryption and flexible broadcast encryption, which combine the decentralized, trustless model of traditional public-key encryption with the efficiency guarantees of broadcast encryption. In the specific case of a flexible broadcast encryption scheme, users generate their own public/private keys and can then post their public key in any public-key directory. Subsequently, a user can encrypt to an *arbitrary* set of user public keys with a ciphertext whose size scales polylogarithmically with the number of public keys in the broadcast set. A distributed broadcast encryption scheme is a more restrictive primitive where each public key is also associated with an index, and one can only encrypt to a set of public keys corresponding to different indices.

In this work, we introduce a generic compiler that takes any distributed broadcast encryption scheme and produces a flexible broadcast encryption scheme. Moreover, whereas existing concretely-efficient constructions of distributed broadcast encryption have public keys whose size scales with the maximum number of users in the system, our resulting flexible broadcast encryption scheme has the appealing property that the size of each public key scales with the size of the maximum broadcast set.

We provide an implementation of the flexible broadcast encryption scheme obtained by applying our compiler to the distributed broadcast encryption scheme of Kolonelos, Malavolta, and Wee (ASIACRYPT 2023). With our scheme, a sender can encrypt a 128-bit symmetric key to a set of over 1000 recipients (from a directory with a million users) with a 2 KB ciphertext. This is 16× smaller than separately encrypting to each user using standard ElGamal encryption. The cost is that the user public keys in flexible broadcast encryption are much larger (50 KB) compared to standard ElGamal public keys (32 bytes). Compared to the similarly-instantiated distributed broadcast encryption scheme, we achieve a 32× reduction in the user's public key size (50 KB vs. 1.6 MB) without changing the ciphertext size. Thus, flexible broadcast encryption provides an efficient way to encrypt messages to large groups of users at the cost of larger individual public keys (relative to vanilla public-key encryption).

1 Introduction

Suppose a user wants to send an encrypted message to K different recipients. The message could be an encrypted email to K recipients or an end-to-end encrypted group chat with K other participants. Using vanilla public-key encryption, the user looks up each recipient's public key in a public-key directory and separately encrypts the message to each recipient. In this case, the encrypted broadcast contains K ciphertexts and its size necessarily scales linearly with the number of recipients K. A natural question is whether we can construct a system where a user can encrypt a message to K different public keys with a *single* ciphertext whose size scales sublinearly with K?

Broadcast encryption. Broadcast encryption [FN93] provides one approach to reduce communication. In a (public-key) broadcast encryption scheme supporting up to L users, each user has an index $i \in [L]$ and a secret key sk_i associated with the index. The encrypter can encrypt a message to a set $S \subseteq [L]$ such that every user $i \in S$ can decrypt using their secret key sk_i. Non-authorized users associated with indices $i \notin S$ cannot decrypt even if they collude. Moreover, the size of the ciphertext scales *sublinearly* with the total number of recipients. Note that a description of the set must be included as part of the ciphertext. In many cases, the set is implicitly known (e.g., "all group members") or admits a short description (e.g., "all computer science students").

While broadcast encryption provides an elegant way for users to encrypt messages to multiple recipients, it comes at the cost of *centralization*. In public-key encryption, users have the ability to generate and manage their own keys, independently of all other users. In contrast, broadcast encryption introduces a central authority that is responsible for *issuing* keys to different users. The central authority represents a central point of failure in a system, and if its secret key is ever compromised, the adversary gains the ability to decrypt *all* ciphertexts in the system. While one could implement the key-generation process for broadcast encryption under multiparty computation (MPC) as a way to avoid generating and storing a long-term master decryption key, this requires all users to join the system at once and jointly participate in the MPC protocol. This is the approach taken in decentralized broadcast encryption [PPS12]. However, the high degree of coordination between users makes such solutions difficult to realize in practice.

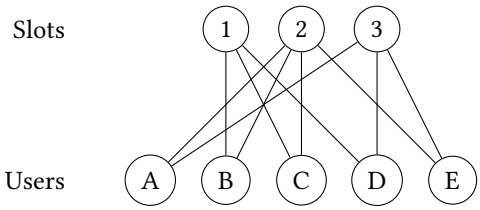
Distributed broadcast encryption. The notion of distributed broadcast encryption [BZ14] or ad-hoc broadcast encryption [WQZD10] interpolates between centralized broadcast encryption and public-key encryption. In this model, each user is still associated with a slot index $i \in [L]$, but the user generates their public/private keys individually (and without coordinating with other users). Each user then posts their public key in a public-key directory (e.g., a public bulletin board). Thereafter, anyone can encrypt a message to any subset of the users with a ciphertext whose size scales sublinearly with the number of recipients.

To decrypt in a distributed broadcast encryption scheme, a user needs to look up the *public* keys of all of the users in the broadcast set. This is a one-time look-up, and in many schemes [KMW23, FWW23], if the broadcast set S is fixed, it is possible to "precompute" a short group-specific public key pk_S associated with the broadcast set that can be cached and reused for many broadcasts (c.f., Remark 4.3).

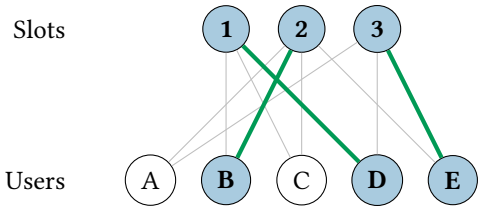
Boneh and Zhandry showed how to construct a distributed broadcast encryption using indistinguishability obfuscation (and one-way functions) [BZ14]. More recently, Kolonelos, Malatova, and Wee [KMW23] as well as Freitag, Waters, and Wu [FWW23] gave constructions using bilinear maps.[1] While these constructions are lightweight, they still have several limitations:

- **Large public keys:** In the existing pairing-based constructions of distributed broadcast encryption [WQZD10, KMW23, FWW23, HLWW23], the size of each user's public key scales with the *maximum* number of users L supported by the scheme. As a result, the size of the public-key directory scales *quadratically* with the total number of users L. Storing even a modestly-sized public-key directory with a million users would require tens of terabytes of storage. For context, the OpenPGP key server currently includes around 3.5 million PGP keys [CNCG+23] while an encrypted messaging platform like WhatsApp has over 2 billion active users [Dea23].
- **Coordination between users:** While users in a distributed broadcast encryption scheme generate their own public keys, the scheme semantics still require some degree of coordination between users. Namely, users should generate keys for *distinct* indices. While nothing prevents multiple users from posting a public key for the *same* index i, the scheme only supports broadcasting to a set of users whose keys occupy *distinct* indices. While we may try to mitigate the need for coordination by having users choose their index randomly when generating the keys, this incurs quadratic overhead. In particular, if N is the number of possible slot indices and L is the maximum size of the public-key directory, we need to set $|N| = \Omega(L^2)$ to ensure that collisions happen with small probability. In existing constructions, the size of each user's public key scales linearly with the number of indices N, so this strategy leads to a construction where user public keys are $O(L^2)$ long and the overall size of the public-key directory is $O(L^3)$.

[1] The Freitag et al. [FWW23] construction is obtained via their construction of distributed broadcast encryption from any registered attribute-based encryption (ABE) scheme and instantiating the registered ABE scheme with a pairing-based construction (e.g., [HLWW23, ZZGQ23]).



(a) **Key generation.** To generate a fresh public key for the flexible broadcast encryption scheme, a user chooses a (random) set of slots for the distributed broadcast encryption scheme and generates a public key for each of the slots. Here, an edge between a user i and a slot j denotes that user i generated a public key for slot index j in the distributed broadcast encryption scheme. In this example, each user generates a key for exactly 2 slots.



(b) **Encryption.** Consider the same bipartite graph as before where the bottom nodes are associated with the users and the top nodes are associated with the slots of the distributed broadcast encryption scheme. An edge exists between user i and slot j if user i generated a public key pk for slot j. To encrypt to a set of users (highlighted in blue), the encrypter looks for a matching between the nodes associated with the users and the slots of the distributed broadcast encryption scheme. If the matching exists, then the user encrypts the message using the distributed broadcast encryption with respect to the public keys associated with the matched edges (shown in green).

Figure 1: An illustration of our generic transformation from a distributed broadcast encryption scheme to a flexible broadcast encryption scheme. In this example, we consider a distributed broadcast encryption scheme with three slots and in this example, a maximum broadcast set size of $K = 3$.

Flexible broadcast encryption. Our starting observation in this work is that in many settings (e.g., encrypted email or encrypted end-to-end messaging), the size of the broadcast set is much smaller than the total number of users in the public-key directory. Thus, we seek a construction where the scheme parameters primarily scale with an (a priori determined) bound on the maximum broadcast size. Moreover, we aim for a construction that does *not* require any coordination between users. Namely, users post their public key to the public-key directory and subsequently, anyone can encrypt a message to *any* subset of public keys in the directory. The size of the ciphertext is polylogarithmic in the number of recipients.

This is the notion of *flexible broadcast encryption* introduced by Freitag, Waters, and Wu [FWW23]. In the same work, they also described a construction from witness encryption. In this work, we describe a general transformation that takes any distributed broadcast encryption and generically transforms it into a flexible broadcast encryption scheme. Combined with previous pairing-based constructions of distributed broadcast encryption [KMW23], we obtain new *concretely-efficient* flexible broadcast encryption schemes from standard pairing-based assumptions.

1.1 Our Contributions

In this work, we introduce a general compiler that takes any distributed broadcast encryption scheme and compiles it into a flexible broadcast encryption scheme. While the parameters of a distributed broadcast encryption scheme can depend *polynomially* on the bound on the total number of users L, the parameters of our flexible broadcast encryption scheme only depends on the size of the maximum broadcast set K. While $K = L$ in the worse case, in settings where $K \ll L$, we achieve considerable savings.

Construction overview. We begin with a high-level overview of our compiler (see also Fig. 1) and refer to Section 3.2 for the full details. For simplicity, we consider a setting with up to $L = 2^\lambda$ users and we want to support broadcasts to a maximum of $K \leq L$ users; here, λ is a (computational) security parameter.

- **Setup:** To support broadcast sets of size at most K, we start with a distributed broadcast encryption scheme with $N = O(K)$ slots (see Theorems 3.4 and 3.5 for the precise requirements).

- **Key generation:** To generate a public key (Fig. 1a), a user selects a random set of $D = O(\lambda)$ indices $S \subseteq [N]$ and generates a public key $\mathsf{pk}^{(i)}$ for each index $i \in S$ in the underlying distributed broadcast encryption scheme. The user's public key is $\{\mathsf{pk}^{(i)}\}_{i \in S}$.
- **Encryption:** To encrypt a message to a set T of K users, the encrypter takes the set of public keys $\{\mathsf{pk}_j\}_{j \in T}$ where $\mathsf{pk}_j = \{\mathsf{pk}_j^{(i)}\}_{i \in S_j}$ for a set $S_j \subseteq [N]$ of size D. The encrypter associates each user $j \in T$ with a *unique* slot index $i_j \in S_j \subseteq [N]$ in the underlying distributed broadcast encryption scheme (see Fig. 1b). It constructs the ciphertext by encrypting to the slot indices $\{i_j\}_{j \in T}$ using the distributed broadcast encryption scheme. The mapping from users j to slot indices i_j can be computed using a bipartite matching algorithm.
- **Decryption:** To decrypt a ciphertext encrypted to a set of public keys $\{\mathsf{pk}_j\}_{j \in T}$, the decrypter first derives the indices $i_j \in S_j$ for each $j \in [T]$. Importantly for correctness, we require that the same set of slot indices is used for both encryption and decryption; we ensure this by using a *deterministic* bipartite matching algorithm in our construction. The user then applies the decryption algorithm for the underlying distributed broadcast encryption scheme.

Implementation and evaluation. In this work, we provide a full implementation and empirical evaluation of our approach. We apply our generic compiler to the pairing-based distributed broadcast encryption scheme by Kolonelos, Malavolta, and Wee [KMW23] to obtain a pairing-based flexible broadcast scheme. Like [KMW23], our scheme requires a one-time sampling of a structured reference string (see Remark 4.1 for more discussion). Importantly, this trusted setup is independent of the users and only needs to be performed once; thereafter, the scheme maintains no long-term secrets (in contrast to traditional centralized broadcast encryption).

We compare the computational and storage costs of implementing a public-key directory based on flexible broadcast encryption with a directory based on distributed broadcast encryption, centralized broadcast encryption, as well as one based on standard ElGamal encryption (here, to encrypt a message to K users, the encrypter would concatenate together K ciphertexts, one for each user). We provide the full evaluation in Section 4, but highlight a few key points here:

- For a public-key directory with 2^{20} users, our scheme can broadcast an encapsulated key to a set of $2^{10} = 1024$ users with a ciphertext of size 2 KB. Each user's public key size is 50 KB in this case. Using vanilla ElGamal encryption for the broadcast would lead to a ciphertext that is 16× larger; however, the tradeoff is that each user's public key is significantly shorter with ElGamal (32 bytes vs. 50 KB). If we were to use distributed broadcast encryption for this setting, a scheme with 2 KB ciphertexts would require user public keys that are over 32× larger (1.6 MB). Using a centralized broadcast encryption [BGW05] would require a master public key that is 6.4 MB (and 2 KB ciphertexts). While this substantially reduces the size of the public-key directory, it comes at the cost of needing to trust a central authority.
- Flexible broadcast encryption supports fast encryption and decryption. Encrypting to 2^{10} users in a directory of 2^{16} users takes just 5.9 ms with our scheme. With traditional ElGamal encryption, preparing 2^{10} ciphertexts would require 94 ms. We incur only a modest timing overhead over distributed (≈1.3×) and centralized broadcast (≈2.6×). Moreover, when the broadcast set S is known in advance (e.g., group messaging applications where the set of recipients is static), the encryption/decryption algorithms can be decomposed into an offline phase that precomputes a public key pk_S for the set S and a fast online phase that only operates on the precomputed key pk_S and the message/ciphertext (see Remark 4.3). Using precomputed keys, the encryption and decryption algorithms typically require just 1-2 ms of computation, irrespective of the size of the broadcast set.

2 Flexible Broadcast Encryption

Notation. Throughout this work, we write λ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n]$ to denote the set of integers $[n] = \{1, \ldots, n\}$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We say a function $f(\lambda)$ is negligible if $f = o(\lambda^{-c})$ for all $c \in \mathbb{N}$; we denote this by writing $f(\lambda) = \mathsf{negl}(\lambda)$.

Flexible broadcast encryption. We now recall the notion of a flexible broadcast encryption scheme from [FWW23]. As described in Section 1, in a flexible broadcast encryption scheme, users can generate a public/private key-pair and post their public key in a public-key directory or public bulletin board. Later on, any user can take any arbitrary subset of public keys $\{\mathsf{pk}_i\}_{i \in S}$ and encrypt a message that can only be decrypted by the users whose public keys pk_i are contained in S. Flexible broadcast encryption generalizes both broadcast encryption [FN93] where a central *trusted* authority is responsible for issuing keys as well as distributed broadcast encryption [BZ14, KMW23] where every user is associated with a unique index and a central *untrusted* authority is responsible for aggregating users' public keys into a single broadcast key. We give the formal definition below:

Definition 2.1 (Flexible Broadcast Encryption [FWW23, adapted])**.** A flexible broadcast encryption scheme is a tuple of efficient algorithm $\Pi_{\mathsf{FBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^K, L) \to \mathsf{pp}$: On input the security parameter λ, a bound K on the maximum broadcast set size, and a bound L on the maximum number of public keys, the setup algorithm outputs a set of public parameters pp. We assume that the public parameters pp include a description of the message space $\mathcal{M}$ for the encryption scheme.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: On input the public parameters pp, the key-generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}) \to b$: On input the public parameters pp and a public key pk, the validity-checking algorithm outputs a bit $b \in \{0, 1\}$.
- $\mathsf{Encrypt}(\mathsf{pp}, S, m) \to \mathsf{ct}$: On input the public parameters pp, a set of public keys S, and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{pp}, S, \mathsf{sk}, \mathsf{ct}) \to m/\perp$: On input the public parameters pp, the set of public keys S associated with the ciphertext, a secret key sk, and a ciphertext ct, the decryption algorithm either outputs a message $m \in \mathcal{M}$ or a special symbol $\perp$ to denote a decryption failure.

Correctness. The correctness requirement for a flexible broadcast encryption scheme says that a user can encrypt a message to *any* set S of K public keys (which may contain maliciously-chosen keys) such that any user who holds an honestly-generated secret key for one of the underlying public keys in S is able to decrypt.

Definition 2.2 (Correctness)**.** A flexible broadcast encryption scheme $\Pi_{\mathsf{FBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is correct if for all security parameters $\lambda \in \mathbb{N}$, all bounds $K \in \mathbb{N}$ and $L \in \mathbb{N}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, 1^K, L)$, all keys $(\mathsf{pk}, \mathsf{sk})$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$, and every collection of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_T)$ where $0 \le T < K$ and $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i) = 1$ for all $i \in [T]$, every message $m \in \mathcal{M}$ (where $\mathcal{M}$ is the message space defined by pp), and setting $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, S, m)$ for $S = \{\mathsf{pk}_i\}_{i \in T} \cup \{\mathsf{pk}\}$,

$$\Pr\left[\mathsf{Decrypt}\left(\mathsf{pp}, S, \mathsf{sk}, \mathsf{ct}\right) = m\right] = 1,$$

where the probability is taken over the randomness of KeyGen and Encrypt. We say the scheme is statistically correct if Definition 2.2 holds with probability $1 - \mathsf{negl}(\lambda)$. In addition, for all pp in the support of $\mathsf{Setup}(1^\lambda, 1^K, L)$ and all $(\mathsf{pk}, \mathsf{sk})$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$, we require

$$\Pr[\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}) = 1] = 1.$$

Security. We consider two notions of security for a flexible broadcast encryption scheme. The standard notion of *adaptive security* says that semantic security holds for a ciphertext encrypted to a set of users that is not under the control of the adversary. In this game, we allow the adversary to learn the secret keys of users not appearing in the challenge ciphertext. We also define a weaker notion of *semi-static security* where we do not allow the adversary to learn the secret keys of any user created by the challenger. Note though that in a flexible broadcast encryption, the adversary can always sample public/secret key-pairs itself. This is the analog of semi-static security in the context of broadcast encryption [GW09] and distributed broadcast encryption [BZ14, KMW23].

Definition 2.3 (Adaptive Security)**.** Let $\Pi_{\mathsf{FBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a flexible broadcast encryption scheme We define the following game which is parameterized by a security parameter $\lambda \in \mathbb{N}$ and a bit $b \in \{0,1\}$:

- **Setup phase:** At the beginning of the game, the adversary chooses the broadcast set size 1^K and the maximum number of public keys 1^L. The challenger replies with the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^K, L)$. The challenger initializes a counter $i \leftarrow 0$ to track the adversary's key-generation queries and a set $C \leftarrow \emptyset$ to track the corrupted public keys.
- **Query phase:** The adversary can now issue two types of queries:
  - **Key-generation query:** In a key-generation query, the challenger first increments the counter $i \leftarrow i + 1$. If $i > L$, the challenger replies with $\perp$. Otherwise, the challenger samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and gives pk_i to $\mathcal{A}$.
  - **Corruption query:** In a corruption query, the adversary specifies an index $j \le i$. In response, the challenger replies with sk_j and adds j to C.
- **Challenge phase:** The adversary now specifies a set $S \subseteq [i]$ of size at most K and two messages $m_0, m_1 \in \mathcal{M}$ (where $\mathcal{M}$ is the message space associated with pp). If $S \cap C \neq \emptyset$, the experiment halts with output 0. Otherwise, the challenger computes the ciphertext $\mathsf{ct}_b \leftarrow \mathsf{Encrypt}(\mathsf{pp}, \{\mathsf{pk}_j\}_{j \in S}, m_b)$ and gives ct_b to $\mathcal{A}$.
- **Output phase:** The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

For an adversary $\mathcal{A}$, we define the advantage $\mathsf{Adv}_{\mathsf{FBE},\mathcal{A}}(\lambda)$ of $\mathcal{A}$ in the flexible broadcast encryption scheme to be

$$\mathsf{Adv}_{\mathsf{FBE},\mathcal{A}}(\lambda) := |\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]|$$

in the above security game (with security parameter λ). We say a flexible broadcast encryption scheme is secure if for all efficient adversaries $\mathcal{A}$, $\mathsf{Adv}_{\mathsf{FBE},\mathcal{A}}(\lambda) \le \mathsf{negl}(\lambda)$.

Definition 2.4 (Semi-Static Security)**.** We say a flexible broadcast encryption scheme Π_{FBE} satisfies semi-static security if it satisfies Definition 2.3, except in the query phase, the adversary is only permitted to issue key-generation queries. In particular, the adversary does not make any corruption queries, so the challenge set $S \subseteq [i]$ is allowed to be *any* subset of the keys generated in response to the challenger's key-generation queries. The remainder of the security experiment is unchanged.

Remark 2.5 (Semi-Static to Adaptive Security)**.** A flexible broadcast encryption scheme that is semi-statically secure (Definition 2.4) can be transformed into one that is adaptively secure (Definition 2.3) in the random oracle model at the cost of a factor of 2× overhead in the ciphertext length and number of public keys required by using a "double encryption" technique [FWW23]; this is the same technique used in the context of centralized broadcast encryption [GW09] and distributed broadcast encryption [BZ14, KMW23].

Remark 2.6 (Post-Challenge Queries)**.** The security definitions for a flexible broadcast encryption scheme do not allow the adversary to make *post-challenge* key-generation or corruption queries. This is without loss of generality, since the adversary can simulate post-challenge key-generation queries itself by running $\mathsf{KeyGen}(\mathsf{pp})$. In the adaptive security game (Definition 2.3), the adversary can "pre-corrupt" all of the keys that do not appear in the challenge set S before entering the challenge query. In this case, there are no additional admissible corruption queries the adversary can make in the post-challenge phase.

Remark 2.7 (Unbounded Public-Key Directory)**.** We say that Definition 2.1 supports an arbitrary polynomial number of public keys if the running time of the Setup algorithm is $\mathsf{poly}(\log L)$. In this case, we can also implicitly define $L = 2^\lambda$. The scheme we construct in this paper (Construction 3.3) supports an arbitrary polynomial of public keys. However, for concrete efficiency, it will be convenient to derive parameters for a more conservative upper bound on the number of users (see Section 4).

Remark 2.8 (Correctness for Invalid Keys)**.** Definition 2.1 requires that a flexible broadcast encryption scheme have a validity-checking algorithm IsValid and the correctness requirement (Definition 2.2) only needs to hold when encrypting to a set of keys that satisfies the IsValid predicate. We can consider an alternative definition where there is no explicit IsValid predicate and correctness should hold when encrypting to any set of public keys (or alternatively, a scheme where IsValid always outputs 1). This is the definition from [FWW23]. We note here that any construction that satisfies Definition 2.1 can be generically transformed into one that satisfies the more general definition by essentially merging IsValid with Encrypt. Namely, the modified encryption algorithm Encrypt′ works as follows:

- $\mathsf{Encrypt}'(\mathsf{pp}, S, m)$: For each public key $\mathsf{pk}_i \in S$, run $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i)$. If the key is invalid, then output the message m in the clear. Otherwise, output $\mathsf{Encrypt}(\mathsf{pp}, S, m)$.

Observe that correctness is preserved since the message m is broadcast in the clear whenever a user encrypts to an invalid key. Moreover, this encryption algorithm does not compromise security because honestly-generated keys are always valid (with respect to IsValid). In this case, the behavior of Encrypt′ and Encrypt is identical when encrypting to a set of uncorrupted users (and the functionality requirements on broadcast encryption preclude any hiding property when the broadcast set includes a corrupted user).

In this work, we separate out the validity-checking algorithm from the encryption algorithm to enable better concrete efficiency. In particular, once a key has been validated, the encrypter does not need to *re-validate* the key every time she encrypts to the particular user. In some settings, there may be external mechanisms that users may use to validate public keys (e.g., key-transparency protocols [MBB+15, Bon16, TD17, CDGM19, MKL+20, MKS+23]). Finally, we also note that the key-validation process can be performed *offline* before the user broadcasts her message. In all of these settings, it is advantageous to be able to have separate algorithms for key validation and for encryption.

Distributed broadcast encryption. In Appendix A, we recall the related notion of distributed broadcast encryption [BZ14]. Here, we provide an informal description. In a distributed broadcast encryption scheme (for L users), users join the system by generating a public/private key associated with a specific slot $i \in [L]$; correspondingly, we associate each user with a unique slot index $i \in [L]$. The master public key of the scheme is the list of the users' public keys. The encrypter can encrypt a message to any subset $S \subseteq [L]$ of the users with a ciphertext whose size scales sublinearly with $|S|$. Like flexible broadcast encryption, there is no central authority or long-term secret key in this system. The key distinction in a flexible broadcast encryption scheme is that there is no notion of slots. Users simply join the system by posting their public key.

3 Constructing Flexible Broadcast Encryption

In this section, we describe how to construct a flexible broadcast encryption scheme from any distributed broadcast encryption scheme. Instantiated with the pairing-based distributed broadcast encryption scheme from [KMW23] (see Construction A.7 for a description of the scheme), this yields an efficient flexible broadcast encryption scheme.

3.1 Graph Theory Background

Our construction relies on some basic combinatoric properties of bipartite graphs. We begin with some basic definitions.

Bipartite graphs and matchings. A bipartite graph $G = (U, V, E)$ consists of two sets of vertices U and V and a set of edges E, where each edge $e \in E$ is a pair of nodes $(u, v) \in U \times V$. A matching $M = (S, \rho)$ on G from U to V consists of a set of nodes $S \subseteq U$ and an *injective* labeling function $\rho\colon S \to V$ such that for all $u \in S$, we have that $(u, \rho(u)) \in E$. We say that $M = (S, \rho)$ is a complete matching if $S = U$ (i.e., every node is matched), and that it is maximal if for every matching $M' = (S', \rho')$ on G from U to V, it holds that $|S| \geq |S'|$. Finally, for a set $S \subseteq U$, we write $\Gamma(S) \subseteq V$ to denote the neighborhood of S: $\Gamma(S) = \{v \in V : \exists (u, v) \in E \wedge u \in S\}$.

Theorem 3.1 (Hall's Marriage Theorem [Hal35])**.** Let $G = (U, V, E)$ be a bipartite graph. Then, G has a complete matching from U to V if and only if for every subset $S \subseteq U$, $|\Gamma(S)| \geq |S|$.

Theorem 3.2 (Hopcroft-Karp [HK71])**.** There exists a *deterministic* algorithm FindMatch that takes as input a bipartite graph $G = (U, V, E)$ and outputs a maximal matching from U to V in time $O\big(|E| \cdot |V|^{1/2}\big)$.

3.2 Flexible Broadcast Encryption Compiler

We now describe our generic compiler that takes any distributed broadcast encryption scheme and compiles it into a flexible broadcast encryption scheme. As described in Section 1.1, to support broadcast sets of size up to K and a maximum of L users, we use a distributed broadcast encryption scheme with $N = N(\lambda, K, L)$ slots, where N is determined by the correctness and security requirements. Each user's public key is in turn associated with a set of $D = D(\lambda, K, L)$ slots of the underlying distributed broadcast encryption scheme. We now give the formal description:

Construction 3.3 (Flexible Broadcast Encryption)**.** Let λ be a security parameter, $K = K(\lambda)$ be a bound on the broadcast set size, and $L = L(\lambda)$ be a bound on the maximum number of public keys in the system. Our construction relies on the following building blocks:

- Let $\Pi_{\mathsf{DBE}} = (\mathsf{DBE.Setup}, \mathsf{DBE.KeyGen}, \mathsf{DBE.IsValid}, \mathsf{DBE.Encrypt}, \mathsf{DBE.Decrypt})$ be a distributed broadcast encryption scheme.
- Let FindMatch be a *deterministic* matching algorithm (c.f., Theorem 3.2).

Let $N = N(\lambda, K, L)$ be a slot parameter and $D = D(\lambda, K, L)$ be a degree parameter (whose values will be determined in the security analysis). We construct a flexible broadcast encryption scheme $\Pi_{\mathsf{FBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Encrypt}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}\big(1^\lambda, 1^K, L\big)$: On input the security parameter λ, a bound on the broadcast set size K, and a bound on the maximum number of public keys in the system L, compute the number of slots $N = N(\lambda, K, L)$ and the degree $D = D(\lambda, K, L)$. Sample public parameters $\mathsf{DBE.pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and output $\mathsf{pp} = (\mathsf{DBE.pp}, N, D)$. The message space $\mathcal{M}$ is that associated with DBE.pp.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the parameters $\mathsf{pp} = (\mathsf{DBE.pp}, N, D)$, sample a set $S \subseteq [N]$ of $|S| = D$ distinct elements. For each $i \in S$, sample a key $(\mathsf{pk}^{(i)}, \mathsf{sk}^{(i)}) \leftarrow \mathsf{DBE.KeyGen}(\mathsf{DBE.pp}, i)$. Output

$$\mathsf{pk} = \left\{(i, \mathsf{pk}^{(i)})\right\}_{i \in S} \quad \text{and} \quad \mathsf{sk} = \left\{(i, \mathsf{sk}^{(i)})\right\}_{i \in S}.$$

- $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk})$: On input the parameters $\mathsf{pp} = (\mathsf{DBE.pp}, N, D)$ and a public key $\mathsf{pk} = \{(i, \mathsf{pk}^{(i)})\}_{i \in S}$, the validity-check algorithm verifies that $\mathsf{DBE.IsValid}(\mathsf{DBE.pp}, \mathsf{pk}^{(i)}, i) = 1$ for all $i \in S$. It additionally checks that $|S| = D$ and outputs 1 if all checks pass. Otherwise, it outputs 0.
- $\mathsf{Encrypt}(\mathsf{pp}, S, m)$: On input the parameters $\mathsf{pp} = (\mathsf{DBE.pp}, N, D)$, a set of public keys $S = \{\mathsf{pk}_j\}_{j \in [T]}$, where $\mathsf{pk}_j = \{(i, \mathsf{pk}_j^{(i)})\}_{i \in S_j}$ for some set $S_j \subseteq [N]$, and a message $m \in \mathcal{M}$, the encryption algorithm constructs a bipartite graph $G = (U, V, E)$ as follows:
  - Let $U = [T]$ and $V = [N]$.
  - For each $j \in [T]$, let $E_j = \big\{(j, v) : v \in S_j\big\}$ and let $E = \bigcup_{j \in [T]} E_j$. In particular, for all $j \in [T]$, $\Gamma(j) = S_j$.

  Let $(M, \rho) \leftarrow \mathsf{FindMatch}(G)$. If $|M| \neq T$, output $\mathsf{ct}' = (0, m)$. Otherwise, output the ciphertext $\mathsf{ct}' = (1, \mathsf{ct}')$ where

$$\mathsf{ct}' \leftarrow \mathsf{DBE.Encrypt}\left(\mathsf{DBE.pp}, \left\{\mathsf{pk}_j^{(\rho(j))}\right\}_{j \in [T]}, \rho(S), m\right),$$

  and $\rho([T]) = \{\rho(t) : t \in [T]\}$.
- $\mathsf{Decrypt}(\mathsf{pp}, S, \mathsf{sk}, \mathsf{ct})$: On input the public parameters $\mathsf{pp} = (\mathsf{DBE.pp}, N, D)$, a set of public keys $S = \{\mathsf{pk}_j\}_{j \in [T]}$ where $\mathsf{pk}_j = \{(i, \mathsf{pk}_j^{(i)})\}_{i \in S_j}$ for some set $S_j \subseteq [N]$, a secret key $\mathsf{sk} = \big\{\big(i, \mathsf{sk}_k^{(i)}\big)\big\}_{i \in S_k}$, and a ciphertext

$\mathsf{ct} = (\beta, \mathsf{ct}')$, the decryption algorithm outputs ct' if $\beta = 0$. Otherwise, it constructs the graph $G = (U, V, E)$ from $\{\mathsf{pk}_j\}_{j\in[T]}$ as described in the Encrypt algorithm. It computes $(M, \rho) \leftarrow \mathsf{FindMatch}(G)$ and outputs

$$\mathsf{DBE.Decrypt}\left(\mathsf{DBE.pp}, \left\{\mathsf{pk}_j^{(\rho(j))}\right\}_{j\in[T]}, \mathsf{sk}_k^{\rho(k)}, \mathsf{ct}', \rho([T]), \rho(k)\right).$$

Theorem 3.4 (Correctness)**.** If Π_{DBE} is correct, then Construction 3.3 is correct.

Proof. Take any security parameter $\lambda \in \mathbb{N}$, broadcast set size $K \in \mathbb{N}$, and bound $L \in \mathbb{N}$ on the number of public keys. Take any pp in the support of $\mathsf{Setup}(1^\lambda, 1^K, L)$ and any key-pair $(\mathsf{pk}, \mathsf{sk})$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$. Let $(\mathsf{pk}_1, \ldots, \mathsf{pk}_T)$ be a collection of $0 \le T < K$ public keys where $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i) = 1$ for all $i \in [T]$. Take any message $m \in \mathcal{M}$ (where $\mathcal{M}$ is the message associated with pp) and let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, S, m)$ for $S = \{\mathsf{pk}_i\}_{i\in[T]} \cup \{\mathsf{pk}\}$. We can write $\mathsf{ct} = (\beta, \mathsf{ct}')$. Consider the output of $\mathsf{Decrypt}(\mathsf{pp}, S, \mathsf{sk}, \mathsf{ct})$. We consider two cases:

- If $\beta = 0$, then $\mathsf{ct}' = m$ and the output of Decrypt is $\mathsf{ct}' = m$.
- If $\beta = 1$, then the output of Decrypt is

  $$\mathsf{DBE.Decrypt}\left(\mathsf{DBE.pp}, \left\{\mathsf{pk}_j^{(\rho(j))}\right\}_{j\in[T]}, \mathsf{sk}_k^{\rho(k)}, \mathsf{ct}', \rho([T]), \rho(k)\right).$$

  Since ρ is a deterministic function of the set S and FindMatch is also deterministic, we can appeal to correctness of Π_{DBE} to conclude that Decrypt also outputs m in this case. □

Theorem 3.5 (Security)**.** Let λ be a computational security parameter. For every efficient semi-static (resp., adaptive) adversary $\mathcal{A}$ for Π_{FBE} that makes up to $L = L(\lambda)$ key-generation queries, there exists an efficient semi-static (resp., adaptive) adversary $\mathcal{B}$ for Π_{DBE} such that

$$\mathsf{Adv}_{\mathsf{FBE},\mathcal{A}}(\lambda) \le \mathsf{Adv}_{\mathsf{DBE},\mathcal{B}}(\lambda) + 2 \cdot \sum_{k\le K} p_k(L, N, D), \tag{3.1}$$

where

$$p_k(L, N, D) = C(L, k) \cdot C(N, k) \cdot \left[\frac{C(k, D)}{C(N, D)}\right]^k. \tag{3.2}$$

Here we write $C(n, k)$ to denote the binomial coefficient $\binom{n}{k}$. In particular, when $N \ge 2eK = O(K)$ and $D = \min(N, \omega(\log\kappa) + \log(e^2LN))$, where κ is a statistical security parameter, we have

$$\mathsf{Adv}_{\mathsf{FBE},\mathcal{A}}(\lambda) \le \mathsf{Adv}_{\mathsf{DBE},\mathcal{B}}(\lambda) + K \cdot \mathsf{negl}(\kappa).$$

Proof. We show the claim for the semi-static case. The adaptive case follows analogously. We start by proving the following claim regarding the existence of a matching in a "random" bipartite graph defined by the KeyGen and Encrypt algorithms. A similar analysis was present in [DGM23] for the setting of batching time-lock puzzles.

Claim 3.6. Let κ be a (statistical) security parameter. Let $K = K(\kappa)$ and $L = L(\kappa)$ be arbitrary functions. Let $N \ge 2eK$ and $N \ge D \ge \omega(\log\kappa) + \log(e^2LN)$. Define a distribution on bipartite graphs $G = (U, V, E)$ as follows:

- Set $U = [L]$ and $V = [N]$.
- For each $u \in U$, sample a random set $S_u \subseteq V$ of D distinct edges (i.e., $S_u \xleftarrow{R} 2^V$ subject to $|S_u| = D$). Let $E_u = \{(u, v) : v \in S_u\}$ and let $E = \bigcup_{u\in U} E_u$.

Then,

$$\Pr[\forall S \subseteq U, |S| \le K : \Gamma(S) \ge |S|] \ge 1 - \sum_{k\le K} p_k(L, N, D),$$

where $p_k(L, N, D)$ is as defined in Eq. (3.2). Moreover, when $N \ge 2eK$ and $D \ge \omega(\log\kappa) + \log(e^2LN)$, then

$$\Pr[\forall S \subseteq U, |S| \le K : \Gamma(S) \ge |S|] \ge 1 - K \cdot \mathsf{negl}(\kappa).$$

In words, with overwhelming probability over the choice of the random bipartite graph (U, V, E), the size of the neighborhood for every subset $S \subseteq U$ with up to K nodes is at least $|S|$.

Proof. We use a union bound. Let $k \le K$ be a size parameter. Fix a subset $S \subseteq U$ of size k, and a subset $T \subseteq V$ of size k. We bound the probability that over the random choice of the edges E, $\Gamma(S) \subseteq T$. Since each S_u is sampled independently, we have that

$$\Pr[\Gamma(S) \subseteq T] = \Pr[\forall u \in S : S_u \subseteq T] = \prod_{u \in S} \frac{C(k,D)}{C(N,D)} = \left[\frac{C(k,D)}{C(N,D)}\right]^k$$

By a union bound over all sets S and T of size k, we have that

$$\begin{aligned}\Pr[\exists S \subseteq U, |S| = k : \Gamma(S) \le |S|] &\le C(L,k) \cdot C(N,k) \cdot \left[\frac{C(k,D)}{C(N,D)}\right]^k \\ &= p_k(L,N,D).\end{aligned}$$

The concrete bound is then obtained by union bounding over each value of $k \le K$. For the asymptotic statement, we use the fact that for positive integers $1 \le k \le n$, it holds that $\left(\frac{n}{k}\right)^k \le C(n,k) \le \left(\frac{en}{k}\right)^k$. Then Eq. (3.2) becomes

$$p_k(L,N,D) = C(L,k) \cdot C(N,k) \cdot \left[\frac{C(k,D)}{C(N,D)}\right]^k \le C(L,k) \cdot C(N,k) \cdot \left(\frac{ek}{N}\right)^{Dk} \le \left(\frac{e^2LN}{k^2}\right)^k \left(\frac{ek}{N}\right)^{Dk}.$$

Since $k \ge 1$, $N \ge 2eK$ and $D \ge \omega(\log\kappa) + \log(e^2LN)$ we now have

$$p_k(L,N,D) \le \left(\frac{e^2LN}{2^D}\right)^k \le 2^{-\omega(\log\kappa)} = \mathsf{negl}(\kappa).$$

The claim now holds by union bounding over all K values of k. □

We now return to the proof of Theorem 3.5. We begin by defining two hybrid experiments, each parameterized by a security parameter λ, and adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$:

- $\mathsf{Hyb}_0^{(b)}(\mathcal{A},\lambda)$: This is the semi-static security experiment (Definitions 2.3 and 2.4).
- $\mathsf{Hyb}_1^{(b)}(\mathcal{A},\lambda)$: Same as $\mathsf{Hyb}_0^{(b)}(\mathcal{A},\lambda)$, except in the challenge phase, after the challenger computes $(M,\rho) \leftarrow \mathsf{FindMatch}(G)$, if $M \ne T$, the challenger halts and the output of the experiment is 0. If $M = T$, then the challenger proceeds as in $\mathsf{Hyb}_0^{(b)}$.

To complete the proof, we bound the statistical distance between $\mathsf{Hyb}_0^{(b)}(\mathcal{A},\lambda)$ and $\mathsf{Hyb}_1^{(b)}(\mathcal{A},\lambda)$ and then show that the distributions $\mathsf{Hyb}_1^{(0)}(\mathcal{A},\lambda)$ and $\mathsf{Hyb}_1^{(1)}(\mathcal{A},\lambda)$ are computationally indistinguishable when Π_{DBE} is secure.

Claim 3.7. For all $b \in \{0,1\}$ and all adversaries $\mathcal{A}$,

$$\left|\Pr\left[\mathsf{Hyb}_0^{(b)}(\mathcal{A},\lambda) = 1\right] - \Pr\left[\mathsf{Hyb}_1^{(b)}(\mathcal{A},\lambda) = 1\right]\right| = \sum_{k \in [K]} p_k(L,N,D).$$

Proof. Suppose adversary $\mathcal{A}$ makes $Q \le L$ key-generation queries in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$. Recall that we set $D = \min(N, \omega(\log(\kappa)) + \log(e^2LN))$ We consider two cases:

- First, suppose that $D = \omega(\log\kappa) + \log(e^2LN)$. By definition, this means that $D \le N$. Then, consider the following graph $\hat{G} = (U,V,E)$:
  - Let $\hat{U} = [Q]$ and $\hat{V} = [N]$.
  - For each $j \in [Q]$, let $S_j \subseteq [N]$ be the slot indices the challenger sampled when answering the j^{th} key-generation query. By construction, $S_j \xleftarrow{R} 2^{[N]}$ subject to the requirement that $|S| = D$. Let $E_j = \{(j,v) : v \in S_j\}$ and let $E = \cup_{j \in [Q]} \hat{S}_j$.

The graph $G = (U, V, E)$ is distributed according to the distribution of Claim 3.6 so with probability at least $1 - \sum_{k \in K} p_k(Q, N, D)$, it holds for all subsets $U' \subseteq U$, $|\Gamma(U')| \geq |U'|$. Consider now the graph $G^* = (U^*, V^*, E^*)$ the challenger constructs when preparing the challenge ciphertext. By construction, G^* is a subgraph of G. Therefore, with at least the same probability as before, for all subsets $U' \subseteq U^*$, $|\Gamma(U')| \geq |U'|$ in G^*. By Theorem 3.1, we conclude that with probability $1 - \sum_{k \in K} p_k(Q, N, D)$, there exists a perfect matching in the graph associated with the challenge ciphertext. In this case, the challenger's behavior in the two experiments is identical. Finally, since $Q \leq L$, it holds that $p_k(L, N, D) \geq p_k(Q, N, D)$, and the claim follows.

- Alternatively, suppose that $D = N$. In this case, the graph $G^* = (U^*, V^*, E^*)$ the challenger constructs when preparing the challenge ciphertext is a *complete* bipartite graph. This means a matching exists with probability 1 and the claim holds. □

Claim 3.8. For all efficient adversaries $\mathcal{A}$, there exists an efficient adversary $\mathcal{B}$ such that

$$\mathsf{Adv}_{\mathsf{DBE},\mathcal{B}}(\lambda) = \left|\Pr\left[\mathsf{Hyb}_1^{(0)}(\mathcal{A}, \lambda) = 1\right] - \Pr\left[\mathsf{Hyb}_1^{(1)}(\mathcal{A}, \lambda) = 1\right]\right|.$$

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ where

$$\left|\Pr\left[\mathsf{Hyb}_1^{(0)}(\mathcal{A}, \lambda) = 1\right] - \Pr\left[\mathsf{Hyb}_1^{(1)}(\mathcal{A}, \lambda) = 1\right]\right| = \varepsilon.$$

We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the distributed broadcast encryption game:

1. Algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ to receive the bound on the maximum broadcast size K and the maximum number of public keys L. Algorithm $\mathcal{B}$ uses K and L to compute the number of slots $N = N(\lambda, K, L)$ and the degree $D = D(\lambda, K, L)$.

2. Algorithm $\mathcal{B}$ forwards the number of slots to the distributed broadcast encryption challenger and receives the public parameters $\mathsf{DBE.pp}$. Algorithm $\mathcal{B}$ gives $\mathsf{pp} = (\mathsf{DBE.pp}, N, D)$ to $\mathcal{A}$.

3. Whenever algorithm $\mathcal{A}$ issues a key-generation query, algorithm $\mathcal{B}$ samples the set S as described in KeyGen, and for each $i \in S$, makes a key-generation query to its challenger on slot i to obtain pk_i. It replies to $\mathcal{A}$ with $\{(i, \mathsf{pk}_i)\}$.

4. When algorithm $\mathcal{A}$ makes a challenge query on a set S and messages m_0, m_1, algorithm $\mathcal{B}$ constructs the graph $G = (U, V, E)$ as described in Encrypt and then constructs the matching $(M, \rho) \leftarrow \mathsf{FindMatch}(G)$. If $M \neq U$, algorithm $\mathcal{B}$ outputs 0 exactly as in $\mathsf{Hyb}_1^{(0)}$ and $\mathsf{Hyb}_1^{(1)}$. Otherwise, it makes a challenge query to the distributed broadcast encryption challenger on the same messages m_0, m_1 and the set of keys identified by $\rho(S)$. The distributed broadcast encryption challenger replies with a ciphertext ct_b which algorithm $\mathcal{B}$ forwards to $\mathcal{A}$.

5. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which algorithm $\mathcal{B}$ also outputs.

By construction if $b = 0$, then the output of $\mathcal{B}$ is distributed according to $\mathsf{Hyb}_1^{(0)}$ and if $b = 1$, the output of $\mathcal{B}$ is distributed according to $\mathsf{Hyb}_1^{(1)}$. Correspondingly, this means that

$$\mathsf{Adv}_{\mathsf{DBE},\mathcal{B}}(\lambda) = \left|\Pr\left[\mathsf{Hyb}_1^{(0)}(\mathcal{A}, \lambda) = 1\right] - \Pr\left[\mathsf{Hyb}_1^{(1)}(\mathcal{A}, \lambda) = 1\right]\right|,$$

and the claim holds. □

Security now follows from Claims 3.7 and 3.8. In particular, recall from Claim 3.6 that $p_k(L, N, D)$ is negligible for the asymptotic setting of the parameters in Theorem 3.5. □

Remark 3.9 (On Outputting the Message in the Clear)**.** It may seem unusual that the encryption algorithm in Construction 3.3 outputs the message in the clear if it fails to find a matching in the graph G. However, as shown in Theorem 3.5 (specifically, Claim 3.6), the probability that no matching exists when encrypting to a set of honest keys is negligible (for suitably-chosen parameters). Thus, this does not impact security of the scheme.

It is possible to adapt our construction in the random oracle model to never broadcast the message in the clear (i.e., if no matching exists, then the encryption algorithm simply outputs $\perp$). While there is no reason to do this if we focus on the correctness and security definitions we consider in this work (and which coincide with the traditional requirements on a centralized broadcast encryption scheme), it is plausible that a future application of flexible broadcast encryption might have security requirements which preclude this behavior. We describe this in Appendix B. However, in the rest of this work, we focus exclusively on Construction 3.3 which satisfies the standard correctness and security properties of (flexible) broadcast encryption.

4 Implementation and Evaluation

In this section, we describe the implementation and evaluation of our flexible broadcast encryption scheme. We obtain our flexible broadcast encryption by applying the generic transformation from Construction 3.3 to the distributed broadcast encryption scheme of Kolonelos, Malavolta, Wee [KMW23]; we refer to Appendix A.1 for a self-contained description of the distributed broadcast encryption scheme from [KMW23]. In our experiments, we provide a comparison of flexible broadcast encryption with the following alternative approaches for broadcasting encrypted messages to a group of K users:

- **Public-key encryption:** The most direct way to encrypt a message to K different users is to encrypt the message under each user's public key. In this case, the encrypted broadcast consists of K ciphertexts. In our comparison, we consider the standard version of elliptic-curve ElGamal public-key encryption [Gam84].[2] When using ElGamal encryption to encrypt a message to *multiple* public keys, we reuse the encryption randomness for efficiency; by a hybrid argument, this does not impact security. Reusing the encryption randomness reduces the size of the encrypted broadcast by a factor of 2. Very briefly, let $\mathbb{G}$ be a group of prime order p and generated by g. In ElGamal encryption, the secret key is a random exponent $s \xleftarrow{R} \mathbb{Z}_p$ and the public key is a group element $h = g^s$. To encrypt a message $\mu \in \mathbb{G}$ to a collection of public keys $(h_1, \ldots, h_K)$, the encrypter samples $r \xleftarrow{R} \mathbb{Z}_p$ and outputs the ciphertext $(g^s, \mu \cdot h_1^s, \ldots, \mu \cdot h_K^s)$. Security relies on the decisional Diffie-Hellman (DDH) assumption in $\mathbb{G}$.

- **Centralized broadcast encryption:** In a centralized broadcast encryption scheme [FN93], a central authority issues secret keys to each user. In this work, we compare against the pairing-based broadcast encryption scheme of Boneh, Gentry, and Waters [BGW05] which has *constant-size* ciphertexts (i.e., each ciphertext consists of two group elements). We specifically consider the instantiation of [BGW05] over *asymmetric* (Type-III) pairing groups from [CHTV22, §3.3]. Following the analysis by Chhatrapati, Hohenberger, Trombo, and Vusirikala [CHTV22, §3.6.5], this is the centralized broadcast scheme with the best concrete efficiency. Security of this construction is based on an asymmetric variant of the bilinear Diffie-Hellman Exponent assumption.

- **Distributed broadcast encryption:** Finally, we also compare the performance against the pairing-based distributed broadcast encryption scheme from [KMW23]. We provide a self-contained description of the scheme in Appendix A.1. As described in Appendix A.1, we make small adjustments to the scheme for better concrete efficiency (i.e., adjusting how ciphertext and secret key components are assigned to the base groups $\mathbb{G}_1$ and $\mathbb{G}_2$). Security of this construction is also based on an asymmetric bilinear Diffie-Hellman Exponent assumption.

Throughout this section, we assume the use of "hybrid encryption" where the broadcast mechanism is used to encrypt a symmetric key and the payload (i.e., the message) is encrypted with the symmetric key using an authenticated encryption scheme. For our comparisons, we thus focus exclusively on the key-encapsulation mechanism (KEM). We provide a summary of the asymptotic parameter sizes and the running times for the different approaches in Table 1.

Alternative constructions of flexible broadcast encryption. In Section 5, we describe two previous approaches for constructing flexible broadcast encryption based on *general-purpose* witness encryption [GGSW13, FWW23] or indistinguishability obfuscation [BGI+01, JLS21, FWW23, HLWW23, FFM+23, DP23]. Both of these techniques require

[2] While RSA-OAEP [RSA78, BR94] is also a common choice for public-key encryption (e.g., in systems like PGP), elliptic curve ElGamal is both faster and has shorter ciphertexts and public keys. Thus, we exclusively use ElGamal encryption as our public-key encryption baseline.

	Broadcast Encryption			
	Flexible	**Distributed**	**Centralized**	**PKE**
CRS size	$O(K)$	$O(L)$	–	–
Directory size	$\tilde{O}(LK)$	$O(L^2)$	$O(K)$	$O(L)$
Ciphertext size	$O(1)$	$O(1)$	$O(1)$	$O(K)$
Setup **time**	$\tilde{O}(K)$	$O(L)$	$O(L)$	–
KeyGen **time**	$\tilde{O}(K)$	$O(L)$	$O(1)$	$O(1)$
Encrypt **time**	$\tilde{O}(K)^\dagger$	$O(K)$	$O(K)$	$O(K)$
Decrypt **time**	$\tilde{O}(K)^\dagger$	$O(K)$	$O(K)$	$O(1)$

† Encryption and decryption rely on computing a perfect matching in the bipartite graph induced by key-generation. While the Hopcroft-Karp algorithm (Theorem 3.2) would yield a worst-case performance of $\tilde{O}(K^{1.5})$ for encryption and decryption, we note that when the underlying graph is *random* (as in our setting), the running time is quasilinear in the size of the graph [Mot89, BMST06] in expectation (where the expectation is taken over the randomness for sampling the graph).

Table 1: Asymptotic comparison of different approaches for broadcasting an encrypted message to a public-key directory. We consider approaches based on our flexible broadcast encryption scheme (Construction 3.3), the distributed broadcast encryption of [KMW23], and the centralized broadcast encryption of [BGW05], as well as the "trivial" broadcast scheme using a vanilla public-key encryption (PKE) scheme. Here, L is the number of users in the public-key directory and K is the size of the broadcast set. We suppress all polynomials in the security parameter, and write $\tilde{O}(\cdot)$ to suppress polylogarithmic factors. For the running-time comparisons, we assume that the algorithms have *random* access to their inputs (i.e., the key-generation, encryption, and decryption algorithms do not necessarily have to read the full input).

extremely heavyweight cryptographic primitives and have not previously been implemented (except in the setting of certain *restricted* functionalities [LMA+16, CMR17]). Since our focus in this work is on *concrete efficiency*, we do not provide comparisons against these schemes in our experimental evaluation.

Remark 4.1 (Structured Reference String)**.** Construction 3.3 inherits many of the properties of the underlying distributed broadcast encryption scheme. Notably, if the underlying distributed broadcast encryption scheme requires a *structured* reference string, the same is true for our scheme. As noted above, we instantiate our compiler with the Kolonelos, Malavolta, and Wee [KMW23] construction which has a linear-size structured reference string. The structured reference string can be sampled using multiparty computation [NRBB22]. Once the reference string has been sampled, we do *not* make additional trust assumptions (i.e., a flexible broadcast encryption does not require persistent, long-term secrets). Moreover, the sampling of the reference string is entirely user-independent (unlike decentralized broadcast encryption [PPS12]). The same reference string can also be reused across multiple independent public-key directories.

Still, a natural question to ask is whether we can construct a distributed broadcast encryption scheme with a transparent setup (i.e., a uniform reference string) or even no reference string at all. This is possible from strong tools like witness encryption [FWW23] and indistinguishability obfuscation [BZ14]. Wu, Qin, Zhang, and Domingo-Ferrer [WQZD10] propose a construction from pairings, albeit without a proof of security; the length of the public keys is also *quadratic* in the number of users (as opposed to linear in the case of [KMW23]). It is an interesting question to construct distributed (or flexible) broadcast encryption with efficiency comparable to our current construction with a transparent setup.

4.1 Implementation and Experimental Setup

We instantiate the cryptographic building blocks underlying our construction as follows:

- **Pairing group:** We instantiate the pairing-based broadcast encryption schemes over the BLS-381 pairing group [BLS02] and use the implementation from the Python `petrelic` library (version 0.1.5) [GL20]. The `petrelic` library is a Python wrapper around the `RELIC` library [AGM+22], which is written in C. The BLS-381 pairing group is asymmetric, and the (serialized) representations of an element of the base groups $\mathbb{G}_1$, $\mathbb{G}_2$, and the target group $\mathbb{G}_T$ are 49 bytes, 97 bytes, and 384 bytes, respectively.
- **ElGamal:** For elliptic-curve ElGamal, we use the `libsodium` library (version 1.0.18) [Den23], which uses Curve25519 for the underlying curve.[3] The size of each group element is 32 bytes.

In all cases, the underlying cryptographic building blocks provide an estimated 128 bits of security.

Parameter selection for flexible broadcast. To construct a flexible broadcast encryption scheme that supports a maximum broadcast size K and up to L registered users, Construction 3.3 relies on a distributed broadcast encryption scheme with N slots and where each user generates public keys for D different slots. The parameters N and D are chosen so as to provide security (Theorem 3.5). In our setting, we choose N and D to provide $\kappa = 40$ bits of *statistical* security and $\lambda = 128$ bits of computational security. Specifically, we choose the parameters N and D so that for every adversary $\mathcal{A}$ for our flexible broadcast encryption scheme, there exists an adversary $\mathcal{B}$ for the underlying distributed broadcast encryption scheme such that Eq. (3.1) in Theorem 3.5 satisfies

$$\mathsf{Adv}_{\mathsf{FBE}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{DBE}}(\mathcal{B}) + 2^{-\kappa}.$$

Concretely, we appeal to Eq. (3.2) and choose N, D such that

$$\sum_{k \leq K} p_k \leq \sum_{k \leq K} C(L,k)C(N,k)\left[\frac{C(k,D)}{C(N,D)}\right]^k \leq 2^{-\kappa}.$$

We exhaustively search over values of N, D that satisfy Section 4.1 and choose the configuration that minimizes the size of each user's public key (and correspondingly, the size of the public-key directory). When we instantiate Construction 3.3 with the distributed broadcast encryption from [KMW23] (see Appendix A.1), the size of each user's public key scales with $O(ND)$.

Bipartite matching. We used the bipartite matching function in SciPy (version 1.11.1). The function implements the Hopcroft–Karp algorithm (Theorem 3.2).

Experimental setup. Our implementation of flexible broadcast encryption scheme (based on the distributed broadcast encryption scheme of [KMW23]) consists of 800 lines of code.[4] We collect our benchmarks on an Amazon EC2 `c6i.xlarge` instance running Ubuntu 22.04. The machine has a 4-core Intel Xeon Platinum 8375C CPU @ 2.90GHz and 8 GB of RAM. We use a single-threaded execution environment for all measurements. We run our code using the Python 3.10.6 interpreter. For our running time measurements, we run each experiment 100 times and report the average running time. When reporting parameter sizes (e.g., public key size and ciphertext size), we compute them *analytically* based on the number of group elements and the measured size of each group element.

Simulated public keys. When measuring the encryption and decryption costs for our flexible broadcast encryption scheme for large broadcast sets (i.e., large values of K), a bottleneck is the cost of generating the K public keys. Since each public key in our scheme has size $\tilde{O}(K)$, generating K public keys requires $\tilde{O}(K^2)$ work and storage. This becomes infeasible on our test machine for large values of K.[5] To benchmark the encryption/decryption costs for large values of K, we replace the actual public keys with simulated keys. Specifically, the public keys in our

[3] Note that `libsodium` is a C library while `petrelic` is a Python library, so some amount of variation in the benchmarks may be due to language-level performance differences. However, we note that `petrelic` is a Python wrapper around a C implementation of the pairing curves, which should reduce some of the variability.

[4] The complete implementation is available here: `https://github.com/RachitG54/FlexBroadcast`.

[5] In practice, users only need to generate and post their own key, so this is not a problem. The challenge in benchmarking is we need to simulate the keys for all K users, which results in the quadratic dependence on K.

flexible broadcast encryption consists of a set of public keys for the underlying distributed broadcast encryption scheme (Construction A.7) [KMW23], and each of these underlying public keys consist of a vector of $O(K)$ *structured* group elements. Instead of computing these group elements individually, we simulate the public key by replacing the group elements with *random* group elements. The running time of the encryption/decryption algorithms only depends on the size of the broadcast set K, so the cost of encryption/decryption with respect to a simulated key should be comparable to the cost with respect to an honestly-generated key. More precisely, for a fixed value of K, the encryption/decryption algorithms always performs the *same* number of group operations. Using simulated keys allows us to provide measurements for considerably larger broadcast sets without having to generate and store public keys for all K users.

To validate the use of simulated keys, we measure the running time of the encryption and decryption algorithms using simulated keys as well as honestly-generated keys. For a directory with $L = 2^{10}$ users, and broadcast sets of size K ranging from $K = 2^5$ to $K = 2^{10}$, we observe that the difference between the running time with real keys and that with simulated keys is at most 2%. For the largest case we compared ($L = K = 2^{10}$), encryption and decryption with real keys required 6.08 ms and 8.26 ms, respectively; using simulated keys, encryption and decryption required 5.99 ms and 8.20 ms, respectively. As mentioned previously, all running times are averaged over a minimum of 100 iterations.

Remark 4.2 (User Key Management)**.** While individual public keys in our system scale quasi-linearly with the size of the broadcast set K, users of the systems do *not* necessarily need to download and store the full public keys of other users. This can yield significant reductions in the communication and storage requirements of the system at the user level. This optimization relies on two special properties of the underlying distributed broadcast encryption scheme [KMW23] we use (see Appendix A.1 for a self-contained description):

- Each user's public key in can be partitioned into a short encryption component and a long decryption component. Notably, the encryption component of each key is a *single* group element (i.e., T_i in Construction A.7). The decryption components are long (scale linearly with the number of users) and essentially consist of "cross-terms." If the user's key is associated with slot i, then their decryption key consists of a cross-term $V_{i,j}$ associated with every other slot $j \neq i$.

- To decrypt a ciphertext encrypted to a set of K users, the user needs to read a single group element from each of the other users' public keys. More precisely, if the user's key is registered to slot i, they need to read the cross-term associated with slot i from every other user's public key. Once again, the user does *not* need to download the entirety of every other user's public key in order to decrypt.

These properties directly yield simplifications to user key management for our flexible broadcast encryption scheme. Recall from Construction 3.3 that a user public key in our flexible broadcast encryption scheme consists of $D = \tilde{O}(1)$ public keys for the underlying distributed broadcast encryption scheme.

- To support encryption, users only need to download and store the "encryption" components of each user's public key. Since each public key in the flexible broadcast encryption scheme consists of D public keys for the underlying distributed broadcast encryption scheme, this means the size of the encryption key for each user is $O(D) = \tilde{O}(1)$. Thus, for a group of L users, the total number of group elements a user needs to download or store to be able to broadcast to any K-subset of the L users is $O(DL) = \tilde{O}(L)$.

- For decryption, our compiler has the property that any one of the user's D distributed broadcast encryption keys could be used to construct the ciphertext. This means the user needs to download or store D group elements from every other user's public key. To decrypt ciphertexts encrypted to any K-subset of a group of L users, each user then needs to download or store a total of $O(D^2L) = \tilde{O}(L)$ group elements.

Observe that in both cases, the user does *not* need to download or store the entirety of the public-key directory (which for L users would have size $\tilde{O}(KL)$) in order to support encryption or decryption. This can yield considerable savings in practice.

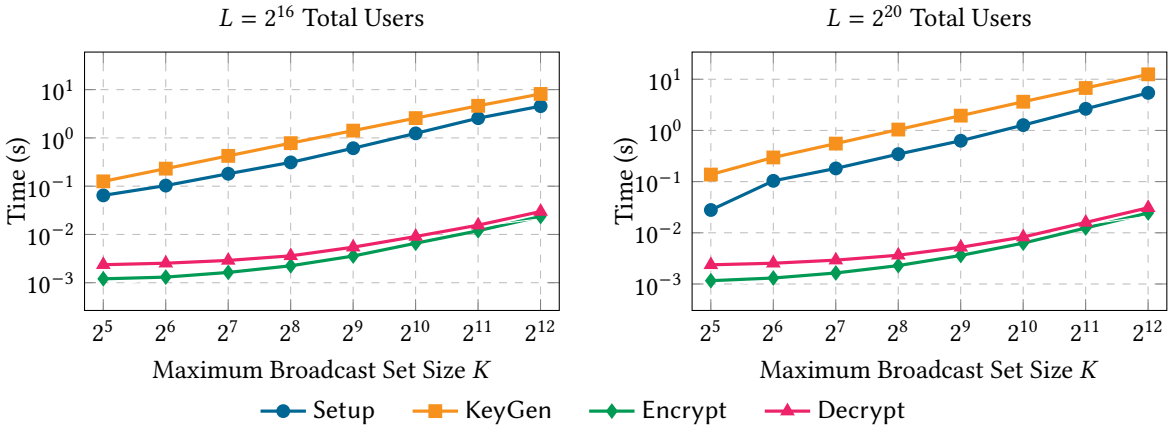


Figure 2: Running time of the primary algorithms for our flexible broadcast encryption scheme (Construction 3.3). We measure the running time of each of the underlying cryptographic algorithms for a public-key directory with $L = 2^{16}$ and $L = 2^{20}$ users and capable of supporting a maximum broadcast set size ranging from $K = 2^5$ to $K = 2^{12}$.

4.2 Benchmarks

In this section, we describe the main benchmarks (in terms of running time and parameter size) for our flexible broadcast encryption scheme and provide a comparison with the alternative approaches for broadcast encryption described at the beginning of Section 4.

Computational cost. Fig. 2 shows the running times of the different cryptographic operations underlying our flexible broadcast scheme as a function of the bound K on the size of the broadcast set. We consider public-key directories with 2^{16} users and 2^{20} users, and measure the performance of each of the cryptographic operations. To support a public-key directory with over a million users, the most expensive operation is user key generation. For broadcast sets with up to 2^{12} users, key generation requires about 12 seconds of computation. While key generation is expensive, this cost is amortized over the lifetime of the key. More importantly, both encryption and decryption are fast. Encrypting to 2^{12} users from a public-key directory with over a million entries requires just 20 ms of computation. For smaller broadcast sets (e.g., 32 users), encryption requires just 2 ms. The decryption time in all of these cases are comparable to the encryption time (since they perform a similar set of operations). Moreover, as we discuss in Remark 4.3, if users send/receive encrypted broadcasts to/from the *same* group of users, they can precompute a group-dependent key to enable even faster encryption and decryption.

In Fig. 3, we compare the computational costs of key generation, encryption, and decryption of flexible broadcast encryption to distributed and centralized broadcast encryption, as well as to that of vanilla ElGamal. Compared to the broadcast encryption schemes, ElGamal encryption has the fastest key generation (over 34000× faster than flexible broadcast) and decryption times (85× faster than flexible broadcast). This is because key-generation and decryption in ElGamal both require a *constant* number of group operations over a *pairing-free* curve. Broadcast encryption schemes all require pairing curves, which incur additional computational overhead. Moreover, the decryption cost for the broadcast encryption schemes all scale linearly with the size of the broadcast set, whereas with ElGamal, a user only has to decrypt the ciphertext encrypted to her public key (see Table 1).

Flexible broadcast encryption yields a 16× reduction in encryption time when encrypting to 2^{10} users. The encryption times among the broadcast encryption schemes are comparable. While the encryption cost for all of the schemes scale linearly (or quasilinearly) with the size of the broadcast set K, the concrete improvement over ElGamal encryption is primarily due to the fact that using ElGamal encryption to encrypt K messages requires $O(K)$ *exponentiations*. In contrast, encryption in the broadcast encryption schemes only require $O(K)$ *multiplications* and a constant number of exponentiations.

When the size of the public key directory coincides with the size of the broadcast set (i.e., $L = K = 2^{10}$), flexible

broadcast encryption is 1.3× slower for encryption and 1.2× slower for decryption (resp., 2.7× and 2.5× slower) compared to distributed (resp., centralized) broadcast encryption. The slowdown relative to distributed broadcast encryption is because our construction (Construction 3.3) instantiates the underlying distributed broadcast encryption scheme with slightly more than K slots in order to reduce the size of each individual public key (see Section 4.1 for details on how we choose the scheme parameters). For instance, for the case of 2^{10} users, our construction initializes the underlying distributed broadcast encryption scheme with 1226 slots and each user generates 4 keys for the underlying scheme.

A key advantage of flexible broadcast over distributed broadcast is the lower key-generation times. Whereas key-generation scales linearly with the size of the public-key directory in distributed broadcast, it scales with the size of the broadcast set in the case of flexible broadcast. When the number of users L is much larger than the size of the broadcast set K (e.g., $L = 2^{24}$ and $K = 2^{10}$), we estimate the key-generation time in flexible broadcast encryption to be 420× faster (7.0 seconds vs. 49 minutes). When the size of the broadcast set is comparable to the total number of users (e.g., $K \approx L$), then distributed broadcast encryption is more efficient. This is because each public key in our flexible broadcast encryption scheme consists of multiple keys for the underlying distributed broadcast encryption scheme; this is what allows us to decouple keys from slot indices. We observe that when the number of users is roughly 16× greater than the size of the broadcast set, flexible broadcast encryption yields faster key generation (see Fig. 3) and shorter public keys.

Setup costs. Each of the broadcast encryption schemes requires an initial setup phase. For centralized broadcast, this corresponds to the central authority generating the master public key and the secret keys for each user. In distributed and flexible broadcast encryption, this is the one-time sampling of a (reusable) common reference string. The advantage of flexible broadcast encryption over the other two schemes is the fact that the setup cost scales with the size of the broadcast set rather than the total number of users in the system (see Table 1).

As a concrete example, when the size of the broadcast set is comparable to the size of the public-key directory, the setup time for flexible broadcast encryption is slightly slower than that for centralized and distributed broadcast (when $K = L = 2^{10}$, the setup for flexible broadcast encryption takes 0.64 s, which is 1.2× slower than that for centralized and distributed broadcast). As the number of users grows (i.e., when $L \gg K$), the setup costs for centralized and distributed broadcast far exceed those for distributed broadcast. For instance, when $L = 2^{14}$ and $K = 2^{10}$, setup for flexible broadcast takes 1.2 s, which is over 7× faster than both distributed and centralized broadcast. This gap further widens as the size of the public-key directory widens.

Public-key directory size. In Fig. 4, we consider the size of the public key directory as a function of the number of users L. As shown in Fig. 4, when L is large, flexible broadcast encryption yields significant savings in the size of the public-key directory compared to distributed broadcast encryption. For instance, to support broadcast sets with up to 64 users, a public-key directory with $L = 2^{16}$ users would require 210 GB of storage with distributed broadcast (i.e., 3.2 MB for a single key). The size of the corresponding directory using flexible broadcast scheme is 5.5 GB (i.e., 83.6 KB for a single key); this is a 38× reduction in storage. This gap widens as the number of users increases as the size of the public-key directory scales *quadratically* with the number of users in the case of distributed broadcast encryption whereas it scales linearly in the case of flexible broadcast encryption (see Table 1).

Both flexible and distributed broadcast require significantly larger public-key directories compared to using a centralized broadcast encryption scheme or vanilla ElGamal encryption. For a public-key directory with the same number of users ($L = 2^{16}$), the size of the public-key directory is just 19 MB in a centralized broadcast encryption scheme and 2.1 MB with vanilla ElGamal public keys. It is an interesting question to design concretely-efficient distributed and flexible broadcast encryption schemes with shorter public keys.

For flexible and distributed broadcast encryption, the public-key directory includes both the CRS as well as the user public keys. The CRS typically constitutes a small fraction of the directory size. For instance, for a directory with $L = 2^{12}$ users and supporting broadcast sets of size $K = 2^6$, the size of the CRS is just 0.018% the size of the public key directory in flexible broadcast encryption. This proportion decreases as the number of users increase.

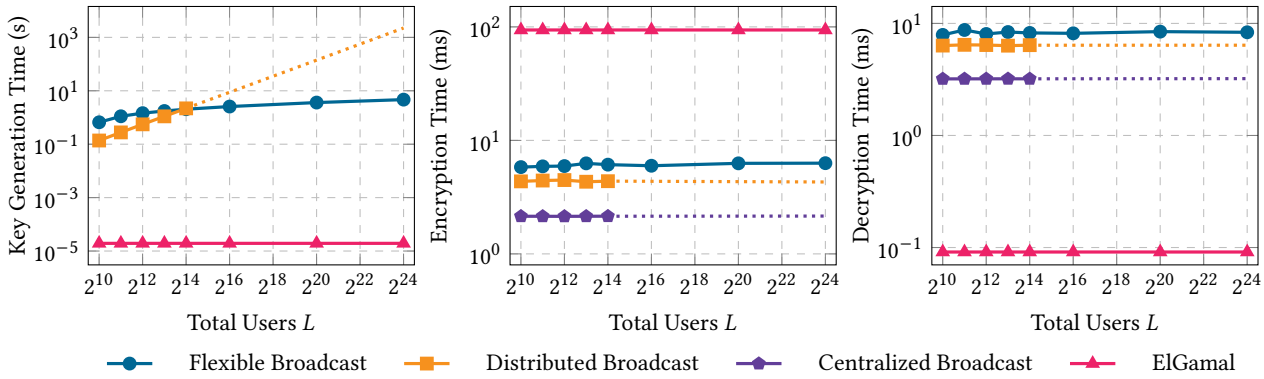


Figure 3: Key-generation (for an individual user), encryption, and decryption times for a broadcast set of size $K = 2^{10}$ in public-key directories of varying sizes L. The solid lines denote *empirically-measured* running times while the dotted lines denote *extrapolated* values based on the measured running times and the asymptotic characterization of the algorithm. We use extrapolated values for distributed broadcast and centralized broadcast in settings where generating the full public-key directory is too expensive. Namely, the generation of the public parameters in centralized broadcast and individual public keys in distributed broadcast both scale linearly with the number of users rather than the size of the broadcast set, making them infeasible to measure for large L. For centralized broadcast encryption, the central authority generates the public/secret key-pairs for the users during setup, so we do not report a user key-generation cost.

Ciphertext size. In Fig. 5, we compare the ciphertext size as a function of the size of the broadcast set. The key advantage of all of the broadcast encryption schemes is that the size of the ciphertext is independent of the number of users in the broadcast set (530 bytes). In contrast, if we have to separately encrypt the message to each user using ElGamal encryption, the size of the ciphertext scales linearly with the number of users. While broadcasting to a single user using ElGamal just requires 64 bytes (8× smaller than broadcast encryption ciphertexts), encrypting to even a modest set of 1024 users already leads to a 33 KB ciphertext. This is over 600× larger than broadcast encryption ciphertexts.

Microbenchmarks. In Fig. 6, we provide a fine-grained breakdown of the computational costs of the encryption and decryption algorithms in our flexible broadcast encryption scheme. Specifically, the encryption and decryption algorithm in Construction 3.3 decomposes into a combinatoric step corresponding to finding a matching in a bipartite graph induced by the broadcast set and a cryptographic step corresponding to constructing a ciphertext (in the case of encryption) or recovering a message (in the case of decryption). As illustrated in Fig. 6, the encryption and decryption costs in our scheme are dominated by the cost of the cryptographic group operations. For example, for a public-key directory with a million users and broadcasting to 64 users, the cryptographic group operations in encryption take 1.3 ms while finding the matching in the bipartite graph requires just 17.2 μs (over 75× faster). However, as the size of the broadcast set increases, the fraction of time taken to compute the matching also increases. When broadcasting to $K = 2^{12}$ users, the combinatoric step accounts for 0.6 ms of the 24 ms required for encryption.

Asymptotically, the number of cryptographic operations in our construction scale linearly with the size K of the broadcast set, whereas the cost of computing the matching scales with $\tilde{O}(K^{1.5})$ in the worst case. For random graphs, such as those induced by Construction 3.3, the (expected) performance of the matching algorithms is *quasilinear* in the size of the graph [Mot89, BMST06]. Ultimately, for the parameter regimes we considered, the running time was dominated by the cryptographic group operations.

Remark 4.3 (Precomputation for Fixed Sets)**.** The distributed broadcast encryption scheme by Kolonelos et al. [KMW23] has an appealing property that the encryption algorithm can be decomposed into two components: (1) a *message-*

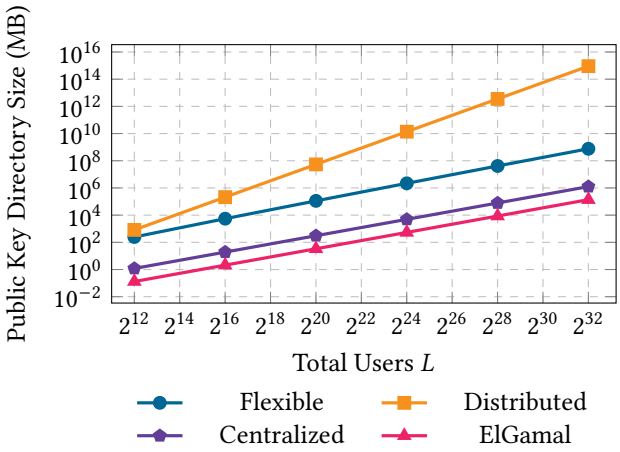


Figure 4: Public-key directory size (computed analytically) needed for different schemes to support broadcasting to a maximum of 2^6 users. We compare the three broadcast encryption schemes: flexible broadcast encryption (Construction 3.3), distributed broadcast encryption [KMW23], and centralized broadcast encryption [BGW05], as well as against a baseline approach of a standard public-key directory with ElGamal public keys. The public-key directory includes the total length of all individual public keys (or the master public key in the case of centralized broadcast) as well as the CRS when applicable. For the settings illustrated here, the CRS for the distributed broadcast encryption constitutes at most 0.2% of the total directory size and 0.02% in the case of flexible broadcast encryption.

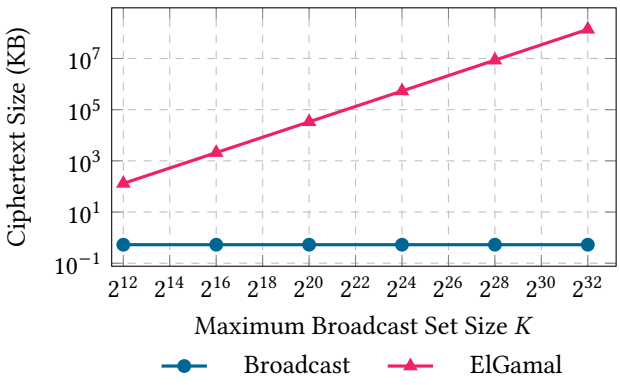


Figure 5: Size of an encrypted broadcast to K users (computed analytically). We compare the ciphertext size using broadcast encryption against the basline approach of separately encrypting to each user using ElGamal encryption. The size of the ciphertext is the same for the centralized broadcast [BGW05], distributed broadcast [KMW23], and our flexible broadcast encryption scheme, so we do not distinguish between these.

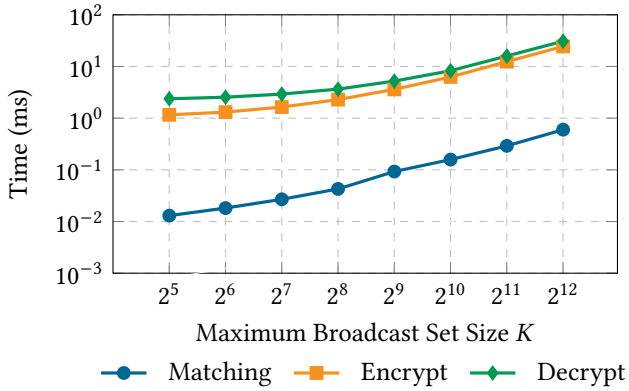


Figure 6: Computational cost breakdown for the encryption and decryption algorithms of our flexible broadcast encryption scheme (Construction 3.3). Both encryption and decryption decompose into (1) a combinatoric step (finding a matching in the bipartite graph induced by the broadcast set); and (2) a cryptographic step for computing the ciphertext (in the case of encryption) or recovering the message (in the case of decryption). We measure the computational costs of constructing the bipartite graph and finding the matching ("Matching") and the cryptographic cost for the encryption ("Encrypt") and decryption algorithms ("Decrypt"). We report the running times as a function of the maximum broadcast set size K. We use a public-key directory with $L = 2^{20}$ users and always assume we are broadcasting to the maximum number of users K.

independent procedure that only depends on the broadcast set S and outputs a short key pk_S; and (2) a *message-dependent* procedure that takes the message m and the precomputed key pk_S, and outputs the ciphertext ct. Critically, given the precomputed key pk_S, the encryption algorithm only needs to perform a constant number of group operations, *independent* of the size of the broadcast set.[6] The decryption algorithm admits a similar decomposition. This means that when encrypting or decrypting to the *same* set of users (e.g., in a group messaging application), users can precompute the public key pk_S for the group members. In this case, the cost of encryption and decryption become *independent* of the size of the broadcast set.

Since our flexible broadcast encryption scheme is built on [KMW23], our scheme also supports precomputing group-dependent public keys to accelerate encryption and decryption. We provide some microbenchmarks in Table 2. Namely, by precomputing the group-dependent public keys, we can reduce the cost of encryption and decryption by 97% and 94%, respectively, when the broadcast set size is $K = 4096$ (i.e., from 23.8 ms to 0.8 ms for encryption and 28.7 ms to 1.7 ms for decryption). This size of the precomputed key is 16.4 KB in this case. Moreover, the improvement increases as the size of the broadcast set increases.

4.3 Ciphertext vs. Directory Size Tradeoff

A limitation of distributed and flexible broadcast encryption is the large size of the users' public keys (and correspondingly, the size of the public-key directory). Whereas a standard ElGamal public key is just 32 bytes, the size of a public key in our flexible broadcast encryption scheme is 14.9 MB (for supporting broadcasts to 2^{14} users). With 2^{20} users, the size of the public-key directory is already 15.6 TB, which can be infeasible to store.

Using a standard rebalancing technique (e.g., [BGW05, §3.2]) we can obtain a tradeoff between the public key size and the ciphertext size in broadcast encryption schemes. For the broadcast encryption schemes we consider in this work, rebalancing allows us to reduce the public parameter size by a factor of N while increasing the ciphertext size by the same factor. We describe the rebalancing approach for each scheme in more detail below:

[6] In the construction of [KMW23] (see Construction A.7), this corresponds to computing $\mathsf{pk}_S = \prod_{i \in S} T_i A_i$, where $\{T_i\}_{i \in S}$ are taken from the public keys of the users in S and $\{A_i\}_{i \in S}$ corresponds to components in the CRS.

	No Precomputation	With Precomputation		
K	**Encryption**	**Online**	**Offline**	**Size**
1024	6 ms	0.8 ms	5.2 ms	4.1 KB
2048	12.1 ms	0.8 ms	11.4 ms	8.2 KB
4096	24.7 ms	0.8 ms	23.8 ms	16.4 KB
	Decryption	**Online**	**Offline**	**Size**
1024	8.1 ms	1.7 ms	6.4 ms	4.1 KB
2048	15.5 ms	1.7 ms	13.8 ms	8.2 KB
4096	30.4 ms	1.7 ms	28.7 ms	16.4 KB

Table 2: Encryption and decryption costs of flexible broadcast encryption with precomputation (Remark 4.3) as a function of the size of the broadcast set K (and a directory with $L = 2^{20}$ users). We report the running time without precomputation, the "online time" of encryption/decryption given the group-dependent key, the "offline time" needed to compute the group-dependent key, and the size of the group-dependent key.

- **Centralized broadcast:** The size of the public parameters in the centralized broadcast scheme from [BGW05] scale linearly with the number of users L. However, the scheme has the appealing property that the *same* public parameters can essentially be *reused* across multiple instantiations (at the cost of publishing one re-randomizing group element for each instantiation) [BGW05, §3.2]. In the rebalanced scheme, to support L users, we partition the L users into N different buckets, where each bucket contains L/N users. We instantiate N copies of the centralized broadcast encryption scheme on L/N users (with a common set of public parameters). Each ciphertext consists of N ciphertexts for the underlying centralized broadcast encryption schemes (where the i^{th} ciphertext is for the L/N users associated with bucket i).
- **Distributed broadcast:** In distributed broadcast, we rebalance by initializing N copies of the distributed broadcast scheme. Each copy supports L/N users and we use the *same* CRS across all of the instantiations. Like in the setting of centralized broadcast encryption, each ciphertext consists of N ciphertexts for the underlying distributed broadcast encryption scheme.
- **Flexible broadcast:** In our flexible scheme, we run N instances of our scheme, where each scheme supports broadcasting to a maximum of K/N users. The *same* CRS is used in all instantiations. Each ciphertext now consists of N ciphertexts for the underlying flexible broadcast encryption scheme, where each ciphertext is used to broadcast to K/N users.

For the rebalanced schemes, the same encryption randomness can be shared across all N ciphertexts (similar to using ElGamal to encrypt the same message to multiple users). We provide an asymptotic comparison of the rebalanced schemes in Table 3.

We illustrate these tradeoffs in Fig. 7. Without rebalancing, the size of a public-key directory with $L = 2^{20}$ users implemented using a flexible broadcast encryption with a maximum broadcast set size $K = 2^{10}$ is 1.3 TB (1.3 MB per key) and each ciphertext is 530 bytes. With rebalancing (i.e., setting $N = 2^5$), we obtain a public key directory of 52 GB (52 KB per key) and a ciphertext size of 2 KB. This represents a 26× reduction in the public key size over the unbalanced scheme at the cost of a 4× increase in the ciphertext size. With rebalancing, the size of the ciphertext is still 16× shorter than using vanilla ElGamal encryption.

5 Related Work

In this section, we survey some related approaches for constructing flexible broadcast encryption and similar primitives, as well as connections between flexible broadcast encryption and other cryptographic protocols.

	Broadcast Encryption			
	Flexible	**Distributed**	**Centralized**	**PKE**
CRS size	$O(K/N)$	$O(L/N)$	–	–
Directory size	$\tilde{O}(LK/N)$	$O(L^2/N)$	$O(N+K/N)$	$O(L)$
Ciphertext size	$O(N)$	$O(N)$	$O(N)$	$O(K)$

Table 3: Asymptotic comparison of different approaches for broadcasting an encrypted message to a public-key directory with rebalancing. Here, L is the number of users in the public-key directory, K is the size of the broadcast set, and N is the rebalancing factor. We suppress all polynomials in the security parameter, and write $\tilde{O}(\cdot)$ to suppress polylogarithmic factors.

Theoretical approaches for constructing flexible broadcast encryption. Flexible broadcast encryption was first introduced by Freitag, Waters, and Wu [FWW23]. In their work, they describe how to realize it by combining witness encryption [GGSW13] with function-binding hash functions. While there have been several recent advancements for constructing witness encryption from new lattice-based assumptions [Tsa22, VWW22], these existing constructions are primarily of theoretical interest currently. To our knowledge, no implementations of these schemes currently exist, and moreover, using them to realize the [FWW23] compiler requires composing witness encryption on top of fully homomorphic encryption (needed to instantiate the function-binding hash function).

In the same work, Freitag et al. also show a generic transformation from a registered attribute-based encryption (registered ABE) scheme [HLWW23] to either a distributed or flexible broadcast encryption scheme, depending on whether the key-generation algorithm in the underlying registered ABE scheme is stateful or stateless. Existing constructions of registered ABE from pairings have a stateful key-generation process [HLWW23] and thus, only yield distributed broadcast encryption, whereas registered ABE schemes based on indistinguishability obfuscation [HLWW23, FFM+23, DP23] are stateless. These latter schemes can be used to construct flexible broadcast encryption, but the reliance on general-purpose indistinguishability obfuscation makes these schemes concretely inefficient.

Dynamic broadcast encryption. The notion of dynamic broadcast encryption [DPP07] was defined as a strengthening of centralized broadcast that allows a (trusted) group manager to dynamically add or remove users to the system while minimally changing the encryption key and without changing users' decryption keys. Subsequently, Phan, Pointcheval, and Strefler [PPS12] extended the notion to a decentralized setting that removes the need for a central authority. However, their scheme requires an interactive setup between all of the users in the system. Flexible broadcast encryption only requires a one-time (reusable) and *user-independent* sampling of the public parameters (see Remark 4.1).

Secure group messaging. Secure group messaging (SGM) protocols [ACDT20, ACDT21, BBR+23] and more formally, continuous group key agreement (CGKA) protocols [ACJM20, AMT23] are cryptographic protocols that facilitate encrypted asynchronous communication between groups of users. The security goals in these protocols are to achieve notions like forward secrecy and post-compromise security. The focus of these works is on supporting end-to-end encrypted messaging among persistent long-lived, and possibly dynamic, groups of users. On the other hand, flexible broadcast encryption is a cryptographic primitive for facilitating short encrypted broadcasts to arbitrary sets of users (and without *a priori* coordination between users).[7]

We view the difference between flexible broadcast encryption and continuous group key agreement to be akin to the difference between vanilla public key encryption and vanilla key agreement. The former allows a user to send an encrypted message without any prior interaction with a recipient whereas the latter allows persistent, long-lived state and may require additional security properties such as forward secrecy. Much like public-key encryption is a

[7] Our construction does require a trusted sampling of a structured reference string (e.g., using a multiparty computation protocol [NRBB22]). However, this sampling procedure is user-independent and the same reference string could be reused across many independent systems. We refer to Remark 4.1 for more discussion.

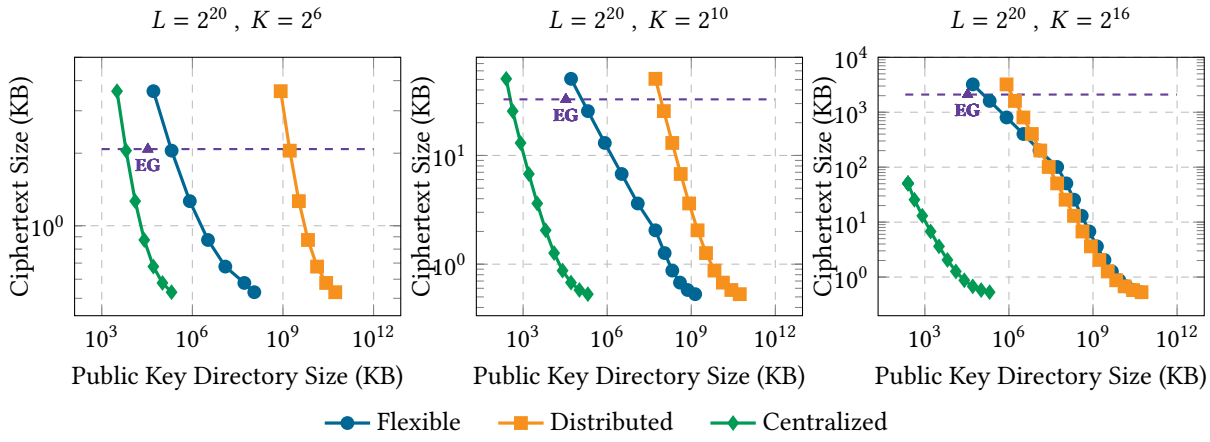


Figure 7: Tradeoff between ciphertext size and public-key directory size (computed analytically) for a public-key directory containing L users and supporting broadcasts to up to K users. We compare our flexible broadcast encryption scheme ("Flexible") against the centralized broadcast encryption scheme of Boneh, Gentry, and Waters [BGW05] ("Centralized") as well as the distributed broadcast encryption scheme of Kolonelos, Malavolta, and Wee [KMW23] ("Distributed"). We instantiate the centralized and distributed broadcast encryption schemes for L users (in order to support broadcasting to an arbitrary subset of K users); as such, both the centralized and the distributed schemes also support broadcasting to more than K users. We also compare against a baseline solutions where the public-key directory consists of ElGamal public keys ("EG") for each user and a broadcast to K users consists of K ElGamal ciphertexts. We showcase these tradeoffs for small ($K = 2^6$), medium ($K = 2^{10}$), and large ($K = 2^{16}$) broadcast sets.

useful building block for realizing key agreement and encrypted messaging, we believe flexible broadcast encryption can be a useful cryptographic building block for improving the efficiency of secure group messaging protocols.

6 Conclusion

Flexible broadcast encryption provides a mechanism to encrypt a broadcast to a set of K users with a ciphertext whose size is sublinear in K. Compared to traditional public-key directories built on vanilla public-key encryption, a flexible broadcast encryption scheme enables succinct broadcasts at the cost of larger individual user keys. This work provides the first concretely-efficient construction of flexible broadcast encryption. With our scheme, a sender can encrypt a 128-bit symmetric key to a set of 2^{10} recipients (from a public-key directory with 2^{20} keys) with a 2 KB ciphertext. This is 16× smaller than separately encrypting to each user using standard ElGamal encryption. This gap widens as the size of the broadcast set increases. The tradeoff is that each user's public key is now 50 KB. While this is relatively short in absolute terms, it is still over 1500× longer than a standard ElGamal public key (32 bytes).

Acknowledgments

We thank Jesko Dujmovic, Giulio Malavolta, and Hoeteck Wee for helpful discussions on this work. We thank the anonymous reviewers for helpful comments on the presentation. Brent Waters is supported by NSF CNS-1908611, CNS-2318701, and a Simons Investigator award. David J. Wu is supported by NSF CNS-2151131, CNS-2140975, CNS-2318701, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[ACDT20] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In *CRYPTO*, 2020.

[ACDT21] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Modular design of secure group messaging protocols and the security of MLS. In *ACM CCS*, 2021.

[ACJM20] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In *TCC*, 2020.

[AGM+22] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. Relic is an efficient library for cryptography. https://github.com/relic-toolkit/relic, 2022.

[AMT23] Joël Alwen, Marta Mularczyk, and Yiannis Tselekounis. Fork-resilient continuous group key agreement. *IACR Cryptol. ePrint Arch.*, 2023.

[BBR+23] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The messaging layer security (MLS) protocol. Technical report, IETF, 3 2023.

[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 *CRYPTO*, 2001.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, 2005.

[BLS02] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *SCN*, 2002.

[BMST06] Hannah Bast, Kurt Mehlhorn, Guido Schäfer, and Hisao Tamaki. Matching algorithms are fast in sparse random graphs. *Theory Comput. Syst.*, 2006.

[Bon16] Joseph Bonneau. Ethiks: Using ethereum to audit a CONIKS key transparency log. In *FC 2016 International Workshops, BITCOIN, VOTING, and WAHC*, 2016.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, 1994.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[CDGM19] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. Seemless: Secure end-to-end encrypted messaging with less</> trust. In *ACM CCS*, 2019.

[CHTV22] Arush Chhatrapati, Susan Hohenberger, James Trombo, and Satyanarayana Vusirikala. A performance evaluation of pairing-based broadcast encryption systems. In *ACNS*, 2022.

[CMR17] Brent Carmer, Alex J. Malozemoff, and Mariana Raykova. 5Gen-C: Multi-input functional encryption and program obfuscation for arithmetic circuits. In *ACM CCS*, 2017.

[CNCG+23] Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J. Wu, and Bryan Ford. Authenticated private information retrieval. In *USENIX Security Symposium*, 2023.

[Dea23] Brian Dean. WhatsApp 2023 user statistics: How many people use WhatsApp?, 2023.

[Den23] Frank Denis. The Sodium cryptography library, 2023.

[DGM23] Jesko Dujmovic, Rachit Garg, and Giulio Malavolta. Efficient batchable time-lock puzzles, 2023. Manuscript.

[DP23] Pratish Datta and Tapas Pal. Registration-based functional encryption. *IACR Cryptol. ePrint Arch.*, 2023.

[DPP07] Cécile Delerablée, Pascal Paillier, and David Pointcheval. Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In *Pairing*, 2007.

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

[FN93] Amos Fiat and Moni Naor. Broadcast encryption. In *CRYPTO*, 1993.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, distributed broadcast, and more. In *CRYPTO*, 2023.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, 1984.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GL20] Laurent Girod and Wouter Lueks. `petrelic`, 2020. https://petrelic.readthedocs.io/en/latest/.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In *EUROCRYPT*, 2009.

[Hal35] Peter Hall. On representatives of subsets. *Journal of The London Mathematical Society*, 1935.

[HK71] John E. Hopcroft and Richard M. Karp. A nˆ5/2 algorithm for maximum matchings in bipartite graphs. In *SWAT*, 1971.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *EUROCRYPT*, 2023.

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

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[LMA+16] Kevin Lewi, Alex J. Malozemoff, Daniel Apon, Brent Carmer, Adam Foltzer, Daniel Wagner, David W. Archer, Dan Boneh, Jonathan Katz, and Mariana Raykova. 5Gen: A framework for prototyping applications using multilinear maps and matrix branching programs. In *ACM CCS*, 2016.

[MBB+15] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: bringing key transparency to end users. In *USENIX Security*, 2015.

[MKL+20] Sarah Meiklejohn, Pavel Kalinnikov, Cindy S. Lin, Martin Hutchinson, Gary Belvin, Mariana Raykova, and Al Cutter. Think global, act local: Gossip and client audits in verifiable data structures. *CoRR*, abs/2011.04551, 2020.

[MKS+23] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean F. Lawlor. Parakeet: Practical key transparency for end-to-end encrypted messaging. In *NDSS*, 2023.

[Mot89] Rajeev Motwani. Expanding graphs and the average-case analysis of algorithms for matchings and related problems. In *STOC*, pages 550–561, 1989.

[NRBB22] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. *IACR Cryptol. ePrint Arch.*, 2022.

[PPS12] Duong Hieu Phan, David Pointcheval, and Mario Strefler. Decentralized dynamic broadcast encryption. In *SCN*, 2012.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2), 1978.

[TD17] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via bitcoin. In *IEEE S&P*, 2017.

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In *CRYPTO*, 2022.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-io from evasive LWE. In *ASIACRYPT*, 2022.

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption. In *ACM CCS*, 2010.

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered ABE via predicate encodings. In *ASIACRYPT*, 2023.

A Distributed Broadcast Encryption

In this section, we recall the formal definition of distributed broadcast encryption scheme from [BZ14, KMW23]. Then, in Appendix A.1, we recall the pairing-based distributed broadcast encryption scheme from [KMW23, §5] based on the bilinear Diffie-Hellman exponent assumption.

Definition A.1 (Distributed Broadcast Encryption [BZ14, KMW23])**.** A distributed broadcast encryption scheme is a tuple of efficient algorithms $\Pi_{\mathsf{DBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{pp}$: On input the security parameter λ and a bound on the number of users N, the setup algorithm outputs the public parameters pp. We assume the public parameters pp include a description of the message space $\mathcal{M}$ for the encryption scheme.
- $\mathsf{KeyGen}(\mathsf{pp}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: On input the public parameters pp and a slot index $i \in [N]$, the key-generation algorithm outputs a public key pk_i and a secret key sk_i.
- $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}, i) \to b$: On input the public parameters pp, a public key pk, and an index $i \in [N]$, the validity-check algorithm outputs a bit $b \in \{0, 1\}$ indicating whether the key is valid or not.
- $\mathsf{Encrypt}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in S}, S, m) \to \mathsf{ct}$: On input the public parameters pp, a set of public keys pk_i associated with indices $i \in S \subseteq [N]$, and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in S}, \mathsf{sk}, \mathsf{ct}, S, k) \to m/\perp$: On input the public parameters pp, a set $S \subseteq [N]$ of public keys pk_i, a secret key sk, a ciphertext ct, and an index $k \in [N]$, the decryption algorithm either outputs a message $m \in \mathcal{M}$ or a special symbol $\perp$ (to denote a decryption failure).

Definition A.2 (Correctness)**.** A distributed broadcast encryption scheme $\Pi_{\mathsf{DBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is correct if for all security parameters $\lambda \in \mathbb{N}$, all bounds $N \in \mathbb{N}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all sets $S \subseteq [N]$, all indices $k \in S$, all $(\mathsf{pk}_k, \mathsf{sk}_k)$ in the support of $\mathsf{KeyGen}(\mathsf{pp}, k)$, all sets of public keys $\{\mathsf{pk}_j\}_{j \in S \setminus \{k\}}$ where $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_j, j) = 1$, and all messages $m \in \mathcal{M}$ (where $\mathcal{M}$ is the message space specified by pp), and taking $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in S}, S, m)$, it follows that

$$\Pr[\mathsf{Decrypt}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in S}, \mathsf{sk}_k, \mathsf{ct}, S, k) = m] = 1.$$

In addition, we require that for all $i \in [N]$ and $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp}, i)$,

$$\Pr[\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i, i) = 1] = 1.$$

Definition A.3 (Adaptive Security)**.** To define adaptive security for a distributed broadcast encryption scheme $\Pi_{\mathsf{DBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Encrypt}, \mathsf{Decrypt})$, we define the following game which is parameterized by a security parameter λ and a bit $b \in \{0,1\}$:

- **Setup phase:** At the beginning of the game, the adversary $\mathcal{A}$ chooses the bound 1^N on the number of users. The challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and gives pp to $\mathcal{A}$. The challenger also initializes a counter $\mathsf{ctr} \leftarrow 0$, a dictionary D, and a set of slot indices $C \leftarrow \emptyset$. Let $\mathcal{M}$ be the message space specified in the public parameters pp.
- **Query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
  - **Key-generation query:** In a key-generation query, the adversary specifies a slot index $i \in [N]$. The challenger responds by incrementing the counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$, sampling $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$ and replies with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$ to $\mathcal{A}$. The challenger adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to D.
  - **Corruption query:** In a corruption query, the adversary specifies an index $c \in [\mathsf{ctr}]$. In response, the challenger looks up the tuple $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[c]$, replies to $\mathcal{A}$ with sk', and adds ctr to C.
- **Challenge phase:** The adversary specifies a broadcast set $S^* \subseteq [\mathsf{ctr}]$ and two messages $m_0, m_1 \in \mathcal{M}$. If $S^* \cap C \neq \emptyset$, the experiment halts with output 0. Otherwise, the challenger computes the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{pp}, \{\mathsf{pk}_c\}_{c \in S^*}, S^*, m_b)$, and sends ct^* to $\mathcal{A}$.
- **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

For an adversary $\mathcal{A}$, we define the advantage $\mathsf{Adv}_{\mathsf{DBE},\mathcal{A}}(\lambda)$ of $\mathcal{A}$ in the distributed broadcast encryption scheme to be

$$\mathsf{Adv}_{\mathsf{DBE},\mathcal{A}}(\lambda) := |\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]|$$

in the above security game (with security parameter λ). We say that Π_{DBE} is secure if for all efficient adversaries $\mathcal{A}$, $\mathsf{Adv}_{\mathsf{DBE}}(\mathcal{A}) \leq \mathsf{negl}(\lambda)$.

Definition A.4 (Semi-Static Security)**.** We say that a distributed broadcast encryption scheme Π_{DBE} is semi-statically secure if it satisfies Definition A.3, except in the query phase, the adversary does not make any corruption queries.

Remark A.5 (Multiple Key-Generation Queries)**.** Our security definitions for a distributed broadcast encryption scheme (Definitions A.3 and A.4) allow an adversary to make multiple key-generation queries for the *same* slot $i \in [N]$, and then adaptively choose (during the challenge phase) which of the honestly-generated keys it would like to use for each of the slots associated with the challenge ciphertext. This is a stronger requirement than the security definition introduced in [KMW23], which only allows the adversary to make a single key-generation query to each slot. However, the stronger security notion that allows the adversary to adaptively choose which key to use for the challenge ciphertext is critical to our transformation from distributed broadcast encryption to flexible broadcast encryption (Section 3). We note that the distributed broadcast encryption scheme from [KMW23, §5] (recalled in Appendix A.1) satisfies this stronger security requirement (namely, the security reduction supports generating arbitrarily many keys for any uncorrupted slot, which is sufficient to argue semi-static security).

A.1 The [KMW23] Distributed Broadcast Encryption Scheme

For completeness, we recall here the distributed broadcast encryption scheme by Kolonelos, Malavolta and Wee [KMW23]. This scheme can be viewed as a distributed version of the centralized broadcast encryption scheme by Boneh, Gentry, and Waters [BGW05]. The distributed broadcast encryption scheme works relies on a prime-order bilinear group, which we define below:

Definition A.6 (Prime-Order Bilinear Group)**.** A prime-order asymmetric bilinear group generator BilinearGroupGen is an efficient algorithm that takes as input the security parameter 1^λ and outputs $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are base groups with generators g_1, g_2, respectively, and $\mathbb{G}_T$ is a target group. Each of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ have prime order $p = 2^{\omega(\log \lambda)}$. Finally, the function $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map (i.e., for all $x, y \in \mathbb{Z}_p$, it holds that $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$). We require that the group operation in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ and the pairing operation e to be efficiently computable.

The Kolonelos-Malavolta-Wee [KMW23] construction. We now recall the distributed broadcast encryption scheme from [KMW23, §5.1]. As noted previously, the [KMW23] construction operates over a prime-order asymmetric group. In asymmetric groups, the representation size of an element of $\mathbb{G}_1$ is smaller than that of an element of $\mathbb{G}_2$ (concretely, for BLS-381, there is a 2× difference). For this reason, we move the "cross-terms" in the [KMW23] construction (i.e., the components $V_{i,j}$ in Construction A.7) from $\mathbb{G}_2$ to $\mathbb{G}_1$. This reduces the size of each user's public key (and consequently, the size of the public-key directory) by a factor of 2 while modestly increasing the size of the ciphertext (from two $\mathbb{G}_1$ elements and one $\mathbb{G}_T$ element to one element in each of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$). We believe this choice is reasonable in practice since the ciphertexts are small (independent of the number of users), whereas the size of the public key directory scales linearly with the number of users.

Construction A.7 (Distributed Broadcast Encryption [KMW23, §5.1])**.** Let BilinearGroupGen be a prime-order asymmetric bilinear group generator. The [KMW23] distributed broadcast encryption scheme $\Pi_{\mathsf{DBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the number of slots N, the setup algorithm starts by sampling $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{BilinearGroupGen}(1^\lambda)$. It then samples a random exponent $\alpha \xleftarrow{R} \mathbb{Z}_p$ and constructs the following encodings of the powers of α:
  - For each $i \in [2N] \setminus \{N+1\}$, let $A_i \leftarrow g_1^{\alpha^i}$.
  - For each $i \in [N]$, let $B_i \leftarrow g_2^{\alpha^i}$.

  Compute $Z = e(A_1, B_N)$ and output the public parameters
  $$\mathsf{pp} = \left(\mathcal{G}, N, Z, \{A_i\}_{i\in[2N]\setminus\{N+1\}}, \{B_i\}_{i\in[N]}\right). \tag{A.1}$$
  The message space $\mathcal{M}$ associated with pp is the target group $\mathcal{M} = \mathbb{G}_T$.
- $\mathsf{KeyGen}(\mathsf{pp}, i)$: On input the public parameters pp (parsed as in Eq. (A.1)) and a slot index $i \in [N]$, the key-generation algorithm samples a random exponent $r_i \xleftarrow{R} \mathbb{Z}_p$ and sets $T_i \leftarrow g_1^{r_i}$. Then, for all $j \in [N]$, it computes the cross-terms $V_{i,j} \leftarrow A_j^{r_i}$. Finally, it outputs
  $$\mathsf{pk}_i = \left(i, T_i, \{V_{i,j}\}_{j\neq N+1-i}\right) \quad \text{and} \quad \mathsf{sk}_i = V_{i,N+1-i}$$
- $\mathsf{IsValid}(\mathsf{pp}, i, \mathsf{pk}_i)$: On input the public parameters pp (parsed as in Eq. (A.1)), a slot index $i \in [N]$, and a public key $\mathsf{pk}_i = \left(i^*, T_i, \{V_{i,j}\}_{j\neq N+1-i}\right)$, the validity-check algorithm outputs 1 if $i = i^*$ and for all $j \neq N+1-i$, it holds that $e(T_i, B_N) = e(V_{i,j}, B_{N-j})$. Otherwise, it outputs 0.
- $\mathsf{Encrypt}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i\in S}, S, m)$: On input the public parameters pp (parsed as in Eq. (A.1)), public keys $\mathsf{pk}_i = \left(i, T_i, \{V_{i,j}\}_{j\neq N+1-i}\right)$ associated with indices $i \in S \subseteq [N]$, and a message $m \in \mathbb{G}_T$, the encryption algorithm starts by sampling a random exponent $s \xleftarrow{R} \mathbb{Z}_p$. It then outputs $\mathsf{ct} = (C_1, C_2, C_3)$ where
  $$C_1 \leftarrow m \cdot Z^s \quad \text{and} \quad C_2 \leftarrow g_2^s \quad \text{and} \quad C_3 \leftarrow \prod_{i\in S} (T_i A_i)^s.$$
- $\mathsf{Decrypt}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i\in S}, \mathsf{sk}, \mathsf{ct}, k)$: On input the public parameters pp (parsed as in Eq. (A.1)), public keys $\mathsf{pk}_i = \left(i, T_i, \{V_{i,j}\}_{j\neq N+1-i}\right)$ associated with indices $i \in S \subseteq [N]$, a secret key $\mathsf{sk} = V^*$, a ciphertext $\mathsf{ct} = (C_1, C_2, C_3)$ and an index $k \in [N]$, the decryption algorithm outputs
  $$C_1 \cdot e\left(V^* \prod_{i\in S\setminus\{k\}} V_{i,N+1-k} \cdot A_{N+1+i-k}, C_2\right) \cdot e\left(C_3^{-1}, B_{N+1-k}\right).$$

Bilinear q-Diffie-Hellman Exponent assumption. Security of the [KMW23] construction follows from the decisional q-Bilinear Diffie-Hellman Exponent (q-BDHE) assumption [BGW05] which we recall below:

Definition A.8 (Decisional q-Bilinear Diffie-Hellman Exponent Assumption [BGW05])**.** Let BilinearGroupGen be a prime-order asymmetric bilinear group generator. For an adversary $\mathcal{A}$, a security parameter λ, and a bit $b \in \{0,1\}$, we define the decisional q-Bilinear Diffie-Hellman Exponent game as follows:

- The challenger starts by sampling a bilinear group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{BilinearGroupGen}(1^\lambda)$.
- Next, the challenger samples exponents $s, \alpha \xleftarrow{R} \mathbb{Z}_p$ and constructs the challenge element $T \in \mathbb{G}_T$ as follows:
  - If $b = 0$, the challenger sets $T \leftarrow e(g_1, g_2)^{s\alpha^{q+1}}$.
  - If $b = 1$, the challenger samples $T \xleftarrow{R} \mathbb{G}_T$.
- The challenger gives the challenge

$$\left(\mathcal{G}, g_1^s, \left\{g_1^{\alpha^i}\right\}_{i\in[2q]\setminus\{q+1\}}, \left\{g_2^{\alpha^i}\right\}_{i\in[q]}, T\right)$$

  to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

The decisional q-Bilinear Diffie-Hellman Exponent (q-BDHE) assumption holds with respect to BilinearGroupGen if for all efficient adversaries $\mathcal{A}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$$

in the above security game.

Theorem A.9 (Distributed Broadcast Encryption from BDHE [KMW23, §5])**.** Construction A.7 is correct. Moreover, if the N-BDHE assumption holds with respect to BilinearGroupGen, then Construction A.7 satisfies semi-static security (Definition A.4).

Remark A.10 (Distributed Broadcast Encryption from k-Linear Assumption)**.** In addition to Construction A.7, Kolonelos [KMW23] also show how to construct an adaptively-secure distributed broadcast encryption scheme from the standard k-linear assumption (for any $k \in \mathbb{N}$) on asymmetric prime-order groups. The disadvantage of this construction is that the size of the public parameters scale *quadratically* with the number of slots, whereas the size of the public parameters Construction A.7 scale linearly. Since the focus of this work is on concrete efficiency, we focus on the more efficient realization that relies on the stronger and less standard q-BDHE assumption.

B Flexible Broadcast Encryption in the Random Oracle Model

As noted in Remark 3.9, it may seem unusual that the encryption algorithm in Construction 3.3 outputs the message in the clear if it fails to find a matching in the graph G. While this does not impact security of the scheme, this property is essential for ensuring *correctness* for *maliciously-chosen* keys. While an honest user generates their public keys for a *random* set of D slots (for the underlying distributed broadcast encryption scheme), a malicious user could construct a public key where the underlying keys for the distributed broadcast encryption scheme all occupy the same set of D slots as an honestly-generated key. When the parameter D is smaller than the size of the broadcast set K, then a malicious user can break correctness by posting a collection of D public keys that use the same slots as the honest key. When encrypting to a set containing the D maliciously-chosen keys and the honestly-generated key, a matching no longer exists. Thus, to ensure the honest user is still able to decrypt, the encryption algorithm simply broadcasts the message in the clear in this case. Broadcasting the message has not compromised security because this event only happens if a user encrypts to a set that contains a corrupted key. In this scenario, the adversary would have learned the message anyways (by decrypting with its own secret key).

It is possible to adapt our construction in the random oracle model to never broadcast the message in the clear (i.e., if no matching exists, then the encryption algorithm simply outputs $\perp$). While there is no reason to do this if we

focus on the correctness and security definitions we consider in this work (and which coincide with the traditional requirements on a centralized broadcast encryption scheme), it is plausible that a future application of flexible broadcast encryption might have security requirements which preclude this behavior. We describe this below:

- To generate a public key, the key-generation algorithm starts by sampling random string $r_1, \ldots, r_D$, where the length of $r_1, \ldots, r_D$ is the number of bits of randomness needed to generate a public key in the underlying distributed broadcast encryption scheme. Let σ be a commitment to $(r_1, \ldots, r_D)$.
- Next, the key-generation algorithm derives the set of slots $S \subseteq [N]$ by hashing the commitment: $S \leftarrow H(\sigma)$. Here, the hash function H is modeled as a random oracle. The key-generation algorithm then samples a public key $(\mathsf{pk}_i, \mathsf{sk}_i)$ for each slot $i \in S$ in the underlying distributed broadcast encryption scheme.
- Finally, the key-generation algorithm provides a non-interactive zero-knowledge (NIZK) proof π that the public keys $\{\mathsf{pk}_i\}_{i \in S}$ were generated honestly using randomness $r_1, \ldots, r_D$ that is associated with the commitment σ.
- The overall public key pk is then $\mathsf{pk} = \left(\sigma, \{\mathsf{pk}_i\}_{i \in S}, \pi\right)$.

The validity-checking algorithm would derive the set S from σ, and then check the NIZK proof that the public keys were correctly-generated (with respect to σ and S). The main observation now is that the slots occupied by a particular public key is now determined by the random oracle. Thus, the only way an adversary can produce a correlated set of slots would be by copying the random oracle input, thus copying the entire key, which we can solve by simply removing duplicate keys in Encrypt. Otherwise, we can appeal to Claim 3.6 to conclude that if the adversary makes at most Q queries to the random oracle, and there are L honest users in the system, then the probability that correctness fails for an honest user (i.e., that exists a subset of up to K public keys for which no matching exists) with probability at most $\sum_{k \le K} p_k(Q + L, N, D)$. If $N \ge 2eK$ and $D = \min(N, \omega(\log \kappa) + \log(e^2(Q + L)N))$, then correctness holds with probability $1 - \mathsf{negl}(\kappa)$.

The above construction imposes a large overhead in the size of the user public keys (due to the inclusion of the NIZK proof). If we consider applying our compiler (Construction 3.3) to a specific distributed broadcast encryption scheme such as the one of Kolonelos et al. [KMW23] (see Construction A.7), we can obtain a more efficient realizations of this idea. We sketch the approach below:

- To generate a key, a user starts by choosing random exponents $r_1, \ldots, r_D \xleftarrow{R} \mathbb{Z}_p$ and computes $T_i \leftarrow g^{r_i}$ for each $i \in [D]$. Next, it derives the slot indices $S \subseteq [N]$ by hashing $T_1, \ldots, T_D$: $S \leftarrow H(T_1, \ldots, T_D)$. As above, we model H as a random oracle.
- Let DBE.pp be the public parameters for an instance of the Kolonelos-Malavolta-Wee distributed broadcast encryption scheme (see Construction A.7). Let $S = (i_1, \ldots, i_D)$. The user now runs $\mathsf{DBE.KeyGen}(\mathsf{DBE.pp}, i_j)$ for each $j \in [D]$; the user uses the element $T_j = g^{r_j}$ as the slot-independent component in the public key (instead of sampling T_j at random).
- Since $T_1, \ldots, T_D$ are part of the user's public key, anyone can verify that the keys were registered to the correct slots.

The key insight in this transformation is that the public keys in the construction of [KMW23] is that each user's public key consists of an *index-independent* component that is independent of the slot index (i.e., the base component T_i) and index-dependent components that depend on the index (i.e., the cross terms $V_{i,j}$). In this case, we can adapt Construction 3.3 to sample the index-independent component of the public key first, and then derive the slot indices by hashing the pre-sampled index-independent component.