Turning HATE Into LOVE: Compact Homomorphic Ad Hoc Threshold Encryption for Scalable MPC

Leonid Reyzin[⋆], Adam Smith[⋆⋆], and Sophia Yakoubov[⋆⋆⋆]

[1] Boston University, {reyzin, ads22}@bu.edu
[2] Aarhus University, sophia.yakoubov@gmail.com

Abstract. In a public-key threshold encryption scheme, the sender produces a single ciphertext, and any $t+1$ out of n intended recipients can combine their partial decryptions to obtain the plaintext. *Ad hoc* threshold encryption (ATE) schemes require no correlated setup, enabling each party to simply generate its own key pair. In this paper, we initiate a systematic study of the possibilities and limitations of ad-hoc threshold encryption, and introduce a key application to scalable multiparty computation (MPC).

Assuming indistinguishability obfuscation (iO), we construct the first ATE that is *sender-compact*—that is, with ciphertext length independent of n. This allows for succinct communication once public keys have been shared. We also show a basic lower bound on the extent of key sharing: every sender-compact scheme requires that recipients of a message know the public keys of other recipients in order to decrypt.

We then demonstrate that threshold encryption that is ad hoc and *homomorphic* can be used to build efficient large-scale fault-tolerant multiparty computation (MPC) on a minimal (star) communication graph. We explore several homomorphic schemes, in particular obtaining one iO-based ATE scheme that is both sender-compact and homomorphic: each recipient can derive what they need for evaluation from a single short ciphertext. In the resulting MPC protocol, once the public keys have been distributed, all parties in the graph except for the central server send and receive only short messages, whose size is independent of the number of participants.

Taken together, our results chart new possibilities for threshold encryption and raise intriguing open questions.

Keywords: Threshold Encryption, Obfuscation, Setup Freeness, Secure Computation

[⋆] Supported in part by NSF grant 1422965.

[⋆⋆] Supported in part by NSF awards IIS-1447700 and AF-1763786 and a Sloan Foundation Research Award.

[⋆⋆⋆] Work done while at Boston University, and was supported in part by NSF grant 1422965.

Table of Contents

1 Introduction

A public key threshold encryption (TE) scheme gives one the ability to generate a ciphertext that is decryptable by any $t+1$ out of n intended recipients, while remaining semantically secure against any smaller group. Among other things, it enables tasks such as electronic voting [HS00,Hir10] and round-efficient multi-party computation (MPC) [MW16,BJMS18], where only $t+1$ colluding parties should be able to learn information about others' inputs.

One simple way to construct threshold encryption is to use any encryption scheme, with each of n recipients having independently generated keys. To encrypt, the sender applies $(t+1,n)$-secret sharing to the message, and encrypts each share with the key of the respective recipient; we call this *share-and-encrypt*.

Share-and-encrypt has the advantage of requiring no master secret and no correlated setup among the recipients. A basic public-key infrastructure is all that is required. We will call TE schemes with this property *ad hoc* threshold encryption (ATE). An additional advantage of this simple approach is that the length of information sent to each recipient is independent of the number of recipients (since each recipient needs to see only the part of the ciphertext relevant to them). We will call TE with this property *recipient-compact*. It is, however, not *sender-compact*, because the length of information sent by the sender is dependent on the number of recipients. This missing feature is particularly desirable when the sender, rather than unicasting information to each recipient, broadcasts it—for example, by using an intermediate server. Prior to this paper, whether sender-compactness is achievable for ad hoc TE was an open problem.

1.1 Our Contributions

In this paper, we initiate a systematic study of the possibilities and limitations of ad hoc threshold encryption, and introduce a key application to scalable MPC. We start with a definitional framework that systematizes the various options for functionality and security in Section 2.

As our main feasibility result (Section 3), we show that sender-compactness is, in principle, achievable.

Contribution 1 (Theorem 1) *We describe the first sender-compact ad hoc threshold encryption scheme.*

The price we pay for sender-compactness is that we use indistinguishability obfuscation (iO), and that every sender needs a public key. This key needs to be known for decryption, and has a component whose size grows polynomially with n. However, public keys are published once, whereas ciphertexts are created and transmitted multiple times, so having the burden of size in the public keys instead of the ciphertexts can be a big advantage when the sender is already known to the recipient. Moreover, in some uses of TE, decryption is delayed, and the linear component of the public key is not needed by every recipient (for example, if TE is used for backup storage that is usually not accessed; see Section 1.2 for another example).

We also show a fundamental limitation of sender-compact schemes: recipients need to know the public keys of other recipients. Specifically, we define (in Section 2) a TE property we call recipient-set-obliviousness, which demands that the recipient algorithms be run without the public keys of other recipients.

Contribution 2 (Theorem 3) *We show that recipient-set-obliviousness and sender-compactness cannot be simultaneously satisfied.*

We formally state and prove this result in Section 4.

Threshold encryption is well suited for applications to multi-party computation (MPC), because it allows multiple parties to learn shares of a value. Building MPC protocols is much easier when encryption also allows for some homomorphic computation, so that operations on unopened ciphertexts can be used to operate on the underlying plaintexts.

We demonstrate, in Section 5, how to build recipient-compact ad hoc threshold encryption schemes that support limited homomorphism. We use the acronym "HATE" to describe ATE schemes that support homomorphism.

Contribution 3 (Theorems 5, 6 and 7) *We describe three recipient-compact HATE schemes that support additive homomorphism.*

The first two of these schemes are based on standard assumptions. They follow the share-and-encrypt paradigm, and allow homomorphism because of a careful combination of specific encryption and secret sharing schemes. One of these schemes keeps messages in the exponent, and thus supports only limited message spaces. Choosing a secret sharing scheme with the right properties is crucial to enable the scheme to be ad hoc, recipient-compact, and homomorphic. We use Shamir and CRT secret sharing, both of which are additively homomorphic over multiple inputs, do not require pre-distributed correlated randomness, and have compact shares.

These schemes are recipient-set-oblivious and therefore cannot be sender-compact, per Section 4. However, they have an additional property on top of recipient compactness, which we call *recipient-local evaluation*: namely, not only does a ciphertext consists of compact recipient-wise components, but also each recipient can perform homomorphic evaluation locally on its own components.

The third recipient-compact additively homomorphic ATE has the advantage that a fresh ciphertext (before homomorphic evaluation) is sender-compact, but at the price of relying on iO. We obtain this scheme by modifying our iO-based scheme from Section 3. Prior to homomorphic evaluation, a different ciphertext (of size independent of n) for each recipient must be extracted from the sender-compact ciphertext. As in the first two schemes, homomorphic evaluation can be performed locally on these per-recipient ciphertexts, giving the scheme *recipient-local evaluation.* This scheme supports only small message spaces.

Open Questions about Ad hoc Threshold Encryption. Our systematic study and results raise several intriguing open problems about ad hoc threshold encryption. First, are there sender-compact ad hoc threshold encryption schemes with constant-size public keys (independent of n)? Are there such schemes which do not require a sender public key? Can such schemes be based on more standard assumptions than iO? Are there ad hoc threshold encryption schemes with ciphertexts that remain compact even after homomorphic evaluation? Is it possible to achieve full homomorphism? (We note that share-and-encrypt is not known to solve this problem: in principle, a multi-input fully homomorphic threshold secret sharing scheme can be combined with fully homomorphic encryption to give fully homomorphic ad hoc threshold encryption; however, to the best of our knowledge, all known constructions of multi-input fully homomorphic threshold secret sharing require pre-distributed correlated randomness.)

The importance of our results and these questions is reinforced by their usefulness for scalable MPC, which we discuss next.

1.2 Application: One-server, Fault-tolerant MPC

Consider a service that has an app with a large smartphone user base. Suppose the service wants to collect aggregate usage statistics, but (for regulatory compliance, or for good publicity, or for fear of becoming a target for attackers and investigators) does not wish to learn the data of any individual user.

A traditional MPC solution is not suitable for this setting, because the phones do not communicate directly with one another, and because we cannot expect every phone to remain engaged for the duration of the protocol, as phones may go out of signal range or run out of charge. We call MPC protocols in this setting Large-scale One-server Vanishing-participants Efficient MPC (LOVE MPC).

As we already mentioned, threshold encryption can be used for MPC. Ad hoc threshold encryption is particularly well-suited for this setting: by not having a setup phase, it eliminates an important bottleneck, because running a multi-user setup protocol with vanishing participants may present problems. In particular, HATE schemes can be used to build LOVE MPC for the honest-but-curious setting as follows: each phone sends an encryption of its input to the server, who homomorphically combines them, sends the result out for decryption by all users, and successfully uses the partial decryptions to get the correct result as long as more than t phones respond.

Using our HATE constructions, we derive a 3-round LOVE MPC for addition (Section 6). This improves on the round complexity of prior work by Bonawitz *et al.* [BIK$^+$17], who proposed a 5-round protocol. (We also prove, in Section 6.1, that three rounds and some setup — e.g. a PKI — is necessary for LOVE MPC.)

The resulting LOVE MPC is based on standard assumptions when using the HATE constructions of Section 5.1, and the linear per-user communication we obtain asymptotically matches the per-user communication of Bonawitz *et al.* [BIK$^+$17]. (Per-user communication was improved to constant by subsequent work of Bell *et al.* [BBG$^+$20], but still at the cost of 5 rounds as opposed to our

3.) Additionally, at the price of using our iO-based HATE construction (Section Section 5.2), we obtain constant per-user communication, which is asymptotically better than the protocol of Bonawitz *et al.* [BIK$^+$17] and asymptotically matches the protocol of Bell *et al.* [BBG$^+$20] (but, of course, at very high concrete costs due to the use of iO).

1.3 Related Work

Threshold Encryption. Known sender-compact threshold encryption schemes are not ad hoc: they require some correlated setup. For instance, a sender-compact threshold variant of ElGamal, due to Desmedt and Frankel [DF90] (and described in Appendix B) requires a setup phase for every new set of n recipients. Delerablée and Pointcheval [DP08] designed a sender-compact scheme based on bilinear maps with a reduced setup requirement. In their scheme, the sender can pick the set of n recipients dynamically; however, each recipient's secret key must be derived from a common master secret key, so this scheme is not ad hoc.

On the other hand, known ad hoc threshold encryption schemes are not sender-compact. The simple share-and-encrypt construction discussed above requires the sender to send an amount of information that is linear in n. Daza *et al.* [DHMR08] use an interpolation-based trick to reduce the ciphertext size to $O(n-t)$ (and subsequently use bilinear maps to give a matching CCA2-secure construction [DHMR07]); however, they leave open the problem of further lowering the ciphertext size.

Ad hoc *fully homomorphic* threshold encryption was explored by Boneh *et al.* [BGG$^+$18] and Badrinarayanan *et al.* [BJMS18], as well as by Dodis *et al.* [DHRW16] as "spooky" encryption; however, their schemes are not even recipient-compact, let alone sender-compact.

Ad Hoc Broadcast Encryption. Ad hoc sender-compact encryption has been achieved in the context of broadcast encryption, which is a special case of threshold encryption with the threshold $t = 0$, giving any one recipient the ability to decrypt. Specifically, Boneh and Zhandry [BZ14] construct what they call *distributed* broadcast encryption form indistinguishability obfuscation (iO). Their construction has the downside of long (polynomial in the number n of recipients) public keys. Later, Ananth et al. [ABG$^+$13] shrink the public keys at the cost of changing the assumption to differing-inputs obfuscation (diO). Zhandry [Zha16] improves on these results, shrinking the public keys and replacing the iO assumption with witness PRFs, but still requiring $t = 0$.

2 Threshold Encryption (TE) Definitions

A threshold encryption scheme [DF90] is an encryption scheme where a message is encrypted to a group $\mathcal{R}$ of recipients, and decryption must be done collaboratively by at least $t + 1$ members of that group. (This can be defined more broadly for general access structures, but we limit ourselves to the threshold access structure in this paper.)

Classically, threshold cryptography involves a secret-shared secret key, which fixes the set of all key-holders. That is, a single Setup operation suffices only to establish a single set of recipients, and the sender is not allowed to specify a recipient set $\mathcal{R}$ at encryption time.

Dynamic threshold encryption [DP08] allows a sender to choose the set of recipients dynamically at encryption time, as described in the Enc algorithm of Section 2.1. In a dynamic threshold encryption scheme, a single Setup operation suffices for the establishment of arbitrarily many groups of recipients.

However, dynamic threshold encryption schemes still require trusted setup, where a central authority distributes correlated randomness to all parties. In an *ad hoc* threshold encryption (ATE) scheme, there is no need for any trusted central authority or master secret key msk. We call a threshold encryption scheme ad hoc if a public-private key pair can be generated without knowledge of a master secret key; that is, if each party is able to generate its keys independently.

In this paper, we additionally consider *keyed-sender* threshold encryption schemes. In a keyed-sender threshold encryption scheme, in order to encrypt a message, the sender must use its own *secret key* in addition to the recipients' public keys (unlike in typical public-key encryption, where encryption does not require the knowledge of any secrets). Similarly, in order to decrypt the ciphertext, recipients need to use the sender's public key in addition to their secret keys.

2.1 Threshold Encryption Syntax

A threshold encryption scheme consists of five algorithms, described in this section. This description is loosely based on the work of Daza *et al.* [DHMR07], but we modify the input and output parameters to focus on those we require in our constructions, with some additional parameters discussed in the text. Parameters in purple (namely, msk) are absent from ad hoc schemes; parameters in blue (namely, sk_{Sndr} and pk_{Sndr}) are present only in keyed-sender schemes (for readers seeing this text in monochrome, we give text explanations in addition to colors). Keyed-sender schemes additionally require a sixth algorithm, $\mathsf{KeyGen}_{\mathsf{Sndr}}$.

$\mathsf{Setup}(1^\lambda, t) \to (\mathsf{params}, \mathsf{msk})$ is a randomized algorithm that takes in a security parameter λ as well as a threshold t and sets up the global public parameters params for the system.
If the scheme is not ad hoc, Setup also sets up the master secret key msk for key generation.
For simplicity, we provide Setup with the threshold t, and assume that t is encoded in params. However, in t-*flexible* schemes, t may be decided by each sender at encryption time, and should then be an input to Enc (and encoded in the resulting ciphertext). In keyed-sender schemes (where the sender must use their secret key to encrypt and recipients must use the sender's public key to decrypt), t may also be specified in the sender's public key.

$\mathsf{KeyGen}(\mathsf{params}, \mathsf{msk}) \to (pk, sk)$ is a randomized key generation algorithm that takes in the global public parameters params (and, if the scheme is not ad hoc, the master secret key msk) and returns a recipient's public-private key pair.

$\mathsf{KeyGen}_{\mathsf{Sndr}}(\mathsf{params}, \mathsf{msk}) \to (pk_{\mathsf{Sndr}}, sk_{\mathsf{Sndr}})$ is a randomized algorithm present in keyed-sender schemes only; it takes in the global public parameters params (and, if the scheme is not ad hoc, the master secret key msk) and returns a sender's public-private key pair where the private key is used to facilitate encryption by the sender, the public key is used to facilitate decryption of messages from the sender.

$\mathsf{Enc}(\mathsf{params}, sk_{\mathsf{Sndr}}, \{pk_i\}_{i\in\mathcal{R},|\mathcal{R}|>t}, m) \to c$ is a randomized encryption algorithm that encrypts a message m to a set of public keys belonging to the parties in the intended recipient set $\mathcal{R}$ in such a way that any size-$(t+1)$ subset of the recipient set should jointly be able to decrypt. We assume t is specified within params, but (if the scheme is keyed-sender) it may also be specified within the sender's public key, or (if the scheme is t-flexible) it may be specified on the fly as an input to Enc itself.

$\mathsf{PartDec}(\mathsf{params}, pk_{\mathsf{Sndr}}, \{pk_i\}_{i\in\mathcal{R}}, sk_j, c) \to d_j$ is an algorithm that uses a secret key sk_j belonging to one of the intended recipients (that is, for $j \in \mathcal{R}$) to get a partial decryption d_j of the ciphertext c. This partial decryption can then be combined with t other partial decryptions to recover the message.

$\mathsf{FinalDec}(\mathsf{params}, pk_{\mathsf{Sndr}}, \{pk_i\}_{i\in\mathcal{R}}, c, \{d_i\}_{i\in\mathcal{R}'\subseteq\mathcal{R},|\mathcal{R}'|>t}) \to m$ is an algorithm that combines $t+1$ or more partial decryptions to recover the message m.

In a sender-compact scheme, the size of the ciphertext c is independent of the number of recipients n. In a recipient-compact scheme, $\mathsf{PartDec}$ requires only a portion c_i of the ciphertext c, where the size of c_i is independent of n.

2.2 Threshold Encryption Flexibility

Not all threshold encryption schemes allow/require all of the algorithm inputs described in Section 2.1. Sometimes disallowing an input can make the scheme less flexible, but, on the other hand, sometimes schemes that do not rely on certain inputs have an advantage.

More Flexibility: Unneeded Inputs. Ad hocness is an example of gaining an advantage by eliminating dependence on an input. Ad hoc schemes do not use the master secret key msk, and thus do not require a trusted central authority (which in many scenarios might not exist).

Another example of gaining an advantage by eliminating an input is *recipient-set-obliviousness.* Requiring both decryption algorithms ($\mathsf{PartDec}$ and $\mathsf{FinalDec}$) to be aware of the set of public keys belonging to individuals in the set $\mathcal{R}$ of recipients can be limiting.

Definition 1 (Threshold Encryption: Recipient-Set-Obliviousness). *We call a threshold encryption scheme* recipient-set-oblivious *if neither partial decryption nor final decryption use* $\{pk_i\}_{i\in\mathcal{R}}$.

It may seem that a recipient-set-oblivious scheme should require less communication, since the sender would never need to communicate $\mathcal{R}$ to the recipients. However, in Section 4 we show that a recipient-set-oblivious ATE scheme cannot be sender-compact.

More Flexibility: Additional Inputs. In describing the threshold encryption algorithms, for the most part we assumed that the threshold t was fixed within the global public parameters params (or, in a keyed-sender scheme, in the sender's public key). However, some schemes (such as share-and-encrypt) allow the sender to choose t at encryption time; we call such schemes *t-flexible*.

2.3 Threshold Encryption Security

The threshold encryption security definition is two-fold. We require *semantic security*, informally meaning that encryptions of two messages of the same size should be indistinguishable. We also require *simulatability*, informally meaning that given a ciphertext corresponding to one of two messages, partial decryptions can be simulated in such a way as to cause the ciphertext to decrypt to either of the two messages. The latter requirement is useful for MPC applications.

Both for semantic security and simulatability, there are three notions of security we consider, which differ according to the point in the security game at which the adversary must commit to the set of corrupt parties $\mathcal{C}$, and the set of challenge ciphertext recipients $\mathcal{R}$. From weakest to strongest, these are *super-static*, *static* and *adaptive* security. In *super-static* security, which is what our obfuscation-based construction achieves, the adversary specifies both $\mathcal{C}$ and $\mathcal{R}$ before seeing the public keys. In *static* security, which is what our other constructions achieve, the adversary specifies $\mathcal{C}$ before seeing the public keys, but can specify $\mathcal{R}$ later, at the same time as the two challenge messages, m_R and m_L. In *adaptive* security, the adversary specifies $\mathcal{C}$ having seen the public keys, and can specify $\mathcal{R}$ at the same time as the two challenge messages, as in static security.

Semantic Security. We use the semantic security definition of Boneh *et al.* [BGG+18] for threshold encryption schemes. Since we are interested in keyed-sender schemes, we need to make two changes to their definition: (a) we need to provide the adversary with the sender public key and use the corresponding sender secret key to encrypt, and (b) in order to get CPA security, we need to allow the adversary to make multiple encryption queries, since encryption is no longer a public operation. In typical public key encryption, a game which allows multiple encryption queries is equivalent to one that does not, but this is not true in a keyed-sender setting. These differences are marked in blue in Figure 1.

Definition 2 (Threshold Encryption: Super Static Semantic Security).

For $b \in \{R, L\}$, let $\mathsf{EXP}(\mathcal{A}, \lambda, u, n, t, b)$ denote the game described in Figure 1 played with the adversary $\mathcal{A}$, security parameter λ, number of existing parties

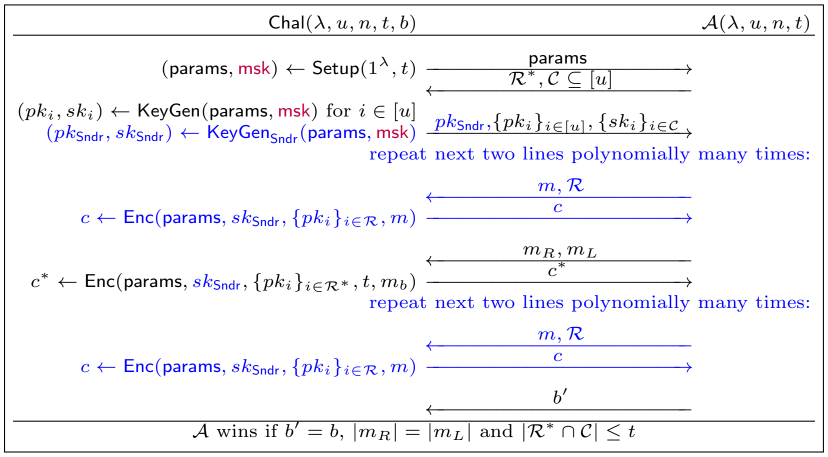
$\mathsf{Chal}(\lambda, u, n, t, b)$
$\mathcal{A}(\lambda, u, n, t)$
$(\mathsf{params}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, t)$
params
$\mathcal{R}^*, \mathcal{C} \subseteq [u]$
$(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\mathsf{params}, \mathsf{msk})$ for $i \in [u]$
$(pk_{\mathsf{Sndr}}, sk_{\mathsf{Sndr}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sndr}}(\mathsf{params}, \mathsf{msk})$
$pk_{\mathsf{Sndr}}, \{pk_i\}_{i \in [u]}, \{sk_i\}_{i \in \mathcal{C}}$
repeat next two lines polynomially many times:
$m, \mathcal{R}$
$c \leftarrow \mathsf{Enc}(\mathsf{params}, sk_{\mathsf{Sndr}}, \{pk_i\}_{i \in \mathcal{R}}, m)$
c
m_R, m_L
$c^* \leftarrow \mathsf{Enc}(\mathsf{params}, sk_{\mathsf{Sndr}}, \{pk_i\}_{i \in \mathcal{R}^*}, t, m_b)$
c^*
repeat next two lines polynomially many times:
$m, \mathcal{R}$
$c \leftarrow \mathsf{Enc}(\mathsf{params}, sk_{\mathsf{Sndr}}, \{pk_i\}_{i \in \mathcal{R}}, m)$
c
b'
$\mathcal{A}$ wins if $b' = b$, $|m_R| = |m_L|$ and $|\mathcal{R}^* \cap \mathcal{C}| \leq t$

Fig. 1: Super Static Semantic Security Game for Threshold Encryption. Objects in blue are only present in keyed-sender schemes; objects in purple are only present in schemes that are not ad hoc.

$|\mathcal{U}| = u$, number of recipients n, threshold t and fixed b. Let $\mathsf{WinProb}(\mathcal{A}, \lambda, u, n, t, b)$ *denote the probability that the adversary $\mathcal{A}$ wins* $\mathsf{EXP}(\mathcal{A}, \lambda, u, n, t, b)$*.*

A threshold encryption scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyGen}_{\mathsf{Sndr}}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{FinalDec})$ *is* (n, t)-super statically semantically secure *if for all $u = poly(\lambda)$ and efficient adversaries $\mathcal{A}$, there exists a negligible function negl such that*

$$|\mathsf{WinProb}(\mathcal{A}, \lambda, u, n, t, R) - \mathsf{WinProb}(\mathcal{A}, \lambda, u, n, t, L)| \leq \frac{1}{2} + negl(\lambda).$$

Note that this definition of security implies that even when the scheme is ad hoc (and therefore KeyGen can be run by participants independently instead of by a trusted central party), KeyGen is assumed to be run honestly; in particular, public keys cannot be generated based on the knowledge of other public keys. We leave the design of definitions and protocols such that public keys *can* be generated maliciously for future work.

Partial Decryption Simulatability. In order to make Definition 2 more analogous to real world situations, it would make sense to additionally allow the adversary to query the challenger on messages of its choice, and receive encryptions of those messages *along with all corresponding partial decryptions.* For the sake of simplicity, instead of modifying the static semantic security game in Figure 1 to include partial decryptions, we add a second notion that we call *partial decryption simulatability* which, informally, implies that receiving partial decryptions will give the adversary no additional information. If a threshold encryption scheme is partial decryption simulatable, then it is possible to simulate remaining partial decryptions given t or fewer partial decryptions, a ciphertext, and a desired plaintext output. Our partial decryption simulatability is similar to,

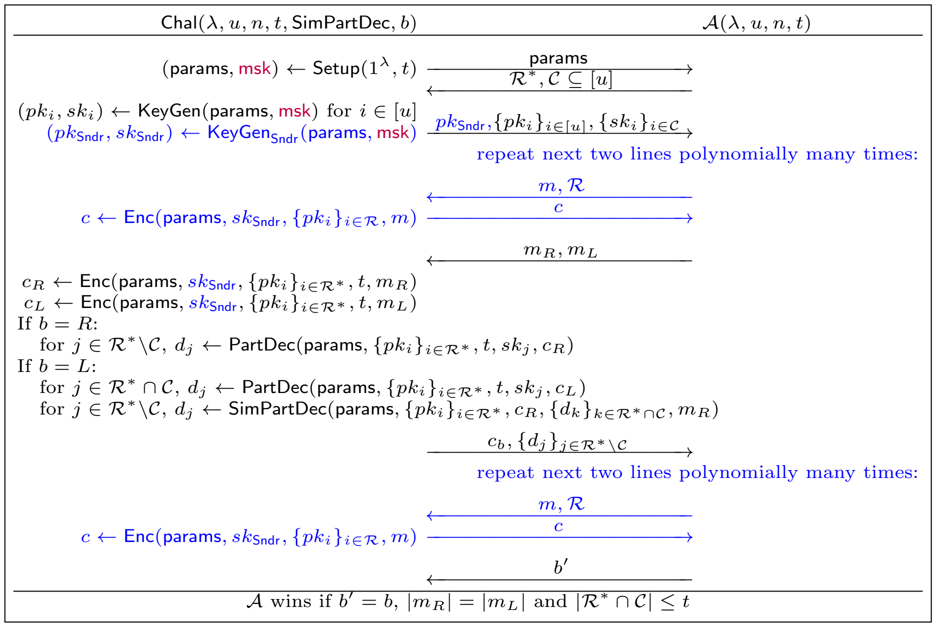
$\mathsf{Chal}(\lambda, u, n, t, \mathsf{SimPartDec}, b)$
$\mathcal{A}(\lambda, u, n, t)$
$(\mathsf{params}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, t)$
params
$\mathcal{R}^*, \mathcal{C} \subseteq [u]$
$(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\mathsf{params}, \mathsf{msk})$ for $i \in [u]$
$(pk_{\mathsf{Sndr}}, sk_{\mathsf{Sndr}}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sndr}}(\mathsf{params}, \mathsf{msk})$
$pk_{\mathsf{Sndr}}, \{pk_i\}_{i \in [u]}, \{sk_i\}_{i \in \mathcal{C}}$
repeat next two lines polynomially many times:
$m, \mathcal{R}$
$c \leftarrow \mathsf{Enc}(\mathsf{params}, sk_{\mathsf{Sndr}}, \{pk_i\}_{i \in \mathcal{R}}, m)$
c
m_R, m_L
$c_R \leftarrow \mathsf{Enc}(\mathsf{params}, sk_{\mathsf{Sndr}}, \{pk_i\}_{i \in \mathcal{R}^*}, t, m_R)$
$c_L \leftarrow \mathsf{Enc}(\mathsf{params}, sk_{\mathsf{Sndr}}, \{pk_i\}_{i \in \mathcal{R}^*}, t, m_L)$
If $b = R$:
for $j \in \mathcal{R}^* \backslash \mathcal{C}$, $d_j \leftarrow \mathsf{PartDec}(\mathsf{params}, \{pk_i\}_{i \in \mathcal{R}^*}, t, sk_j, c_R)$
If $b = L$:
for $j \in \mathcal{R}^* \cap \mathcal{C}$, $d_j \leftarrow \mathsf{PartDec}(\mathsf{params}, \{pk_i\}_{i \in \mathcal{R}^*}, t, sk_j, c_L)$
for $j \in \mathcal{R}^* \backslash \mathcal{C}$, $d_j \leftarrow \mathsf{SimPartDec}(\mathsf{params}, \{pk_i\}_{i \in \mathcal{R}^*}, c_R, \{d_k\}_{k \in \mathcal{R}^* \cap \mathcal{C}}, m_R)$
$c_b, \{d_j\}_{j \in \mathcal{R}^* \backslash \mathcal{C}}$
repeat next two lines polynomially many times:
$m, \mathcal{R}$
$c \leftarrow \mathsf{Enc}(\mathsf{params}, sk_{\mathsf{Sndr}}, \{pk_i\}_{i \in \mathcal{R}}, m)$
c
b'
$\mathcal{A}$ wins if $b' = b$, $|m_R| = |m_L|$ and $|\mathcal{R}^* \cap \mathcal{C}| \leq t$

Fig. 2: Super Static Partial Decryption Simulatability Game for Threshold Encryption. Objects in blue are only present in keyed-sender schemes; objects in purple are only present in schemes that are not ad hoc.

but stronger than, simulatability of partial decryption defined in [MW16], where only a single partial decryption can be simulated.

Definition 3 (Threshold Encryption: Super Static Simulatability). *For $b \in \{R, L\}$, let* $\mathsf{EXP}(\mathcal{A}, \lambda, u, n, t, \mathsf{SimPartDec}, b)$ *denote the game described in Figure 2 played with the adversary $\mathcal{A}$, security parameter λ, number of existing parties $|\mathcal{U}| = u$, number of recipients n, threshold t, simulation algorithm* $\mathsf{SimPartDec}$ *and fixed b.*

Let $\mathsf{WinProb}(\mathcal{A}, \lambda, u, n, t, \mathsf{SimPartDec}, b)$ *denote the probability that the adversary $\mathcal{A}$ wins* $\mathsf{EXP}(\mathcal{A}, \lambda, u, n, t, \mathsf{SimPartDec}, b)$.

A threshold encryption scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyGen}_{\mathsf{Sndr}}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{FinalDec})$ *is* (n, t)-super statically partial decryption simulatable *if there exists an efficient algorithm* $\mathsf{SimPartDec}$ *such that if for all $u = poly(\lambda)$ and efficient adversaries $\mathcal{A}$, there exists a negligible function negl such that*

$$|\mathsf{WinProb}(\mathcal{A}, \lambda, n, t, \mathsf{SimPartDec}, R) - \mathsf{WinProb}(\mathcal{A}, \lambda, n, t, \mathsf{SimPartDec}, L)| \leq \frac{1}{2} + negl(\lambda).$$

Putting it all together, we say that a threshold encryption scheme has super static security if it meets both of the above definitions, as specified in Definition 4.

Definition 4 (Threshold Encryption: Super Static Security). *A threshold encryption scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{FinalDec})$ *is* (n, t)-super statically secure *if it is both* (n, t)-super statically semantically secure *(Definition 2) and* (n, t)-super statically partial decryption simulatable *(Definition 3).*

2.4 Threshold Encryption with Homomorphism

Homomorphic ad hoc threshold encryption (HATE) can be particularly useful in applications to multi-party computation.

Definition 5 (Threshold Encryption: Homomorphism). *Let* $\mathcal{F}$ *be a class of functions, each taking a sequence of valid messages and returning a valid message. An* $\mathcal{F}$-homomorphic *threshold encryption scheme additionally has the following algorithm:*

$\mathsf{Eval}(\mathsf{params}, \{pk_i\}_{i\in\mathcal{R}}, [c_1, \dots, c_\ell], f) \to c^*$ *is an algorithm that, given* ℓ *ciphertexts and a function* $f \in \mathcal{F}$*, computes a new ciphertext* c^* *which decrypts to* $f(m_1, \dots, m_\ell)$ *where each* c_q*,* $q \in [1, \dots, \ell]$ *decrypts to* m_q*.*

Informally, Eval should be *correct*, meaning that decryption should lead to the correct plaintext message $f(m_1, \dots, m_\ell)$.

2.5 Threshold Encryption Compactness

Compactness Without Homomorphism. As described in the introduction, we say that a threshold encryption scheme is *sender-compact* (or, in other words, that it has *sender-compact encryption*) if the size of a ciphertext is independent of the number of recipients. We say that it is *recipient-compact* (or, in other words, that it has *recipient-compact encryption*) if the portion of the ciphertext required by each recipient to produce their partial decryption is independent of the number of recipients. Of course, if a threshold encryption scheme is sender-compact, then it is also recipient-compact, since each receiver can use the entire (compact) ciphertext to partially decrypt. However, the converse is not necessarily true. Even if a scheme is not sender-compact, it can be recipient-compact if the ciphertext c can be split into compact components $c = \{c_i\}_{i\in\mathcal{R}}$ such that every recipient can run $\mathsf{PartDec}$ given just one component c_i.

Compactness With Homomorphism. When we consider homomorphic threshold encryption, a fresh ciphertext c may look different than a ciphertext c^* which Eval outputs. Of course, the size of c^* should not grow linearly with the number ℓ of inputs to f; otherwise, homomorphism becomes unnecessary, and c^* could simply consist of a concatenation of the input ciphertexts.

Notice that this does not preclude ciphertext growth. Even if a fresh ciphertext has size independent of n, the output of Eval may grow with n. We introduce some new terminology to handle this: we say that a homomorphic threshold encryption scheme has *compact evaluation* if the output of Eval has size independent of n, and that it has *recipient-compact evaluation* if the output of Eval can be split into recipient-wise compact components. Additionally, we say that a homomorphic threshold encryption scheme has *recipient-local evaluation* if it has *recipient-compact encryption* and evaluation is performed component-wise, with Eval taking one recipient's component of each input ciphertext and producing that recipient's compact component of the output ciphertext.

All of our schemes in Section 5 have recipient-compact encryption and recipient-local evaluation; the scheme in Section 5.2) additionally has sender-compact encryption.

In a setting where multiple senders send ciphertexts to a single server, who homomorphically computes on the ciphertexts and sends (the relevant parts of) the output of Eval to receivers, it is enough to have a sender-compact encryption and recipient-compact evaluation, even if the overall output of Eval is long. These properties suffice for reducing bandwidth, because the size of every message transmitted between two parties is independent of the number of recipients.

If, instead, we have a setting where senders send ciphertexts directly to receivers who then compute on those ciphertexts themselves, sender-compact encryption is less important, and recipient-local evaluation becomes key. Each sender must send something to each receiver anyway (instead of sending only one thing to the server), and in a setting with direct peer-to-peer channels, it becomes unimportant whether those things are all the same sender-compact ciphertext, or receiver-wise components of a recipient-compact ciphertext.

3 Sender-Compact Ad Hoc Threshold Encryption

In this section, we describe a sender-compact ATE. In the share-and-encrypt construction, the total ciphertext size is $\Theta(n)$, because each recipient gets an encryption of a different share. A natural approach is to compress the ciphertext using obfuscation: namely, instead of using the encrypted shares as the ciphertext, we can try to use an obfuscated program that *outputs* one encrypted share at a time given an appropriate input (such as a short symmetric encryption of the message, a recipient secret key, and proof of the recipient membership in the recipient set $\mathcal{R}$).

However, this strategy fails to achieve sender-compact ciphertexts, because the obfuscated program remains linear in the size of the threshold t. The reason is that, within the security proof, in one of the hybrids we are forced to hardcode t secret shares in the program, and the obfuscated program must be of the same size in all hybrid games.

Therefore, instead of putting an obfuscated program in the ciphertext, each sender obfuscates a program as part of key generation. This program becomes the sender's public key. While it is long (polynomial in the in the number of recipients n), it needs to be created and disseminated only once, as opposed to a ciphertext, which depends on the message. Notice that having this obfuscated program as the sender's public key makes our ATE scheme *keyed-sender*, meaning that in order to encrypt a message the sender must use its secret key, and in order to decrypt a message, recipients must use the sender's public key.

One can think of the obfuscated program in the sender's public key as a "horcrux".[3] The sender stores some of its secrets in this obfuscated program, and when encrypting a message, the sender includes just enough information in the ciphertext that the obfuscated program can do the rest of the work.

Once we put the obfuscated program in the sender's public key, we run into the issue that the outputs of the program on the challenge ciphertext cannot be dependent on the challenge message. This is because in the proof of security, the challenge message is chosen dynamically by the adversary, whereas the program is obfuscated by the challenger at the beginning of the game. In some hybrids, the outputs corresponding to the challenge message must be hardcoded in the program; so, they cannot depend on the actual message, which can be picked after the program is fixed. Therefore, instead of returning secret shares of the challenge message, the program returns shares of a random mask which is used to encrypt the message.

Specifically, the program that each sender obfuscates takes as input a random nonce — together with the sender's signature on that nonce — and a recipient's secret key. The program checks the signature, and that the recipient's secret key matches one of the public keys to which the sender addressed this ciphertext (this "addressing" is performed implicitly, via the same signature). Note that checking membership in the set of recipients is important: otherwise any party could extract a secret share of the message. If the checks pass, the program outputs a secret share of a PRF output on the random nonce. The actual message is symmetrically encrypted with that PRF output.

The obfuscated program that makes up the sender public key is formally described in Algorithm 1, and the obfuscation-based ATE is described in Construction 1. It uses an indistinguishability obfuscator iO, puncturable pseudorandom function PPRF, a secret sharing scheme SS, a constrained signature SIG, and a length-doubling pseudorandom generator PRG with domain $\{0,1\}^\lambda$ and range in $\{0,1\}^{2\lambda}$. We define all of these primitives in Appendix A.

[3] A "horcrux" is a piece of one's soul stored in an external object, according to the fantasy series Harry Potter [Row05].

Algorithm 1 $f_{k_w, k_{\mathsf{Share}}, \mathsf{SIG}.pk}(\overrightarrow{pv} = \{pv_i\}_{i \in \mathcal{R}}, \mathsf{idx}, sv, \mathsf{nonce}, \sigma)$

```
The following values are hardcoded:
    $\mathsf{params} = (\lambda, n, t)$, where
        $\lambda$ is the security parameter,
        $n$ is the number of recipients, and
        $t$ is the threshold.
    $k_w$, a secret PPRF key used to recover the mask $w$ from nonce $\mathsf{nonce}$
    $k_{\mathsf{Share}}$, a secret PPRF key used to secret share the mask $w$
    $\mathsf{SIG}.pk$, a signature verification key
The following values are expected as input:
    $\overrightarrow{pv} = \{pv_i \in \{0,1\}^{2\lambda}\}_{i \in \mathcal{R}}$, lexicographically ordered public values
    idx, an index
    $sv \in \{0,1\}^{\lambda}$, a secret value
    nonce
    $\sigma$, a signature
if ($\overrightarrow{pv}[\mathsf{idx}] = \mathsf{PRG}(sv)$) and ($\mathsf{SIG.Verify}(\mathsf{SIG}.pk, (\overrightarrow{pv}, \mathsf{nonce}), \sigma)$) then
    $w \leftarrow \mathsf{PPRF}_{k_w}(\mathsf{nonce})$
    $r \leftarrow \mathsf{PPRF}_{k_{\mathsf{Share}}}(\mathsf{nonce})$
    $[w]_{\mathsf{idx}} \leftarrow \mathsf{SS.Share}(w, n, t; r)[\mathsf{idx}]$ {This gives the idxth secret share of $w$}
    return $[w]_{\mathsf{idx}}$
```

Informally, in order to prove security, we will have to show that given an obfuscation of this program, an adversary who has only t or fewer secret keys from the recipient set will not be able to tell the difference between an encryption of a message m_R and an encryption of a different message m_L. Our proof will need to puncture k_w and k_{Share} on the challenge nonce in order to remove any information about the challenge plaintext from the program. For the proof to go through given the guarantees of iO, it is crucial that, as we change the plaintext, the output does not change for any input — in particular, even if the adversary is able to forge a signature that ties the ciphertext to a wrong set of public keys. We ensure this property by using a constrained signature scheme SIG, so that we can guarantee (in an appropriate hybrid) that a signature tying the ciphertext to a wrong set of public keys does not exist. This means that the public verification key (which is incorporated into the obfuscated program) is of size polynomial in n.

Theorem 1. *The obfuscation-based ATE (Construction 1) is (n,t)-super-statically secure (Definition 4) for any polynomial n, t, as long as* iO *is a secure indistinguishability obfuscator,* PPRF *is a secure puncturable* PRF*,* SS *is a secure (n,t)-secret sharing scheme,* SIG *is a constrained signature scheme, and* PRG *is a secure pseudorandom generator.*

We prove Theorem 1 in Appendix C.

```
Let the public parameters params = $(\lambda, n, t)$ consist of the security parameter $\lambda$,
the number of recipients $n$, and the threshold $t$.

KeyGen(params):
    {The following generates the "receiver" keys.}
    $sv \leftarrow \{0,1\}^\lambda$
    $pv \leftarrow \mathsf{PRG}(sv) \in \{0,1\}^{2\lambda}$
    return $(pv, sv)$
KeyGen$_{\mathsf{Sndr}}$(params):
    {The following generates the "sender" keys.}
    $(\mathsf{SIG}.pk, \mathsf{SIG}.sk) \leftarrow \mathsf{SIG.KeyGen}(1^\lambda)$
    $k_w \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda)$
    {This PPRF key will be used to produces the mask $w$ for the message. Its output is
    assumed to be in the message space group.}
    $k_{\mathsf{Share}} \leftarrow \mathsf{PPRF.KeyGen}(1^\lambda)$
    {This PPRF key will be used to produce the randomness for secret sharing $w$. We slightly
    abuse PPRF notation above; the size of $w$ and the size of the randomness needed to
    secret share $w$ might be very different. We simply assume that either the keys used
    are of different sizes (that is, $k_{\mathsf{Share}}$ might actually consist of multiple keys), or that
    the PPRF is chained in the appropriate way to produce a sufficiently large amount of
    randomness. We assume that the output of PPRF with $k_w$ is in some group $\mathcal{G}$ which
    contains the message space, and that the output of PPRF with $k_{\mathsf{Share}}$ is of whatever form
    the randomness for SS.Share should take.}
    $\mathsf{ObfFunc} \leftarrow \mathsf{iO}(f_{k_w, k_{\mathsf{Share}}, \mathsf{SIG}.pk})$
    return $(pk_{\mathsf{Sndr}} = \mathsf{ObfFunc}, sk_{\mathsf{Sndr}} = (\mathsf{SIG}.sk, k_w))$
Enc(params, $sk_{\mathsf{Sndr}} = (\mathsf{SIG}.sk, k_w), \overrightarrow{pv} = \{pv_i\}_{i \in \mathcal{R}, |\mathcal{R}| \geq t}, m$):
    $\mathsf{nonce} \leftarrow \mathsf{PPRF.domain}$
    $e = (\mathsf{PPRF}_{k_w}(\mathsf{nonce}) + m)$
    $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{SIG}.sk, (\overrightarrow{pv}, \mathsf{nonce}))$
    return $c = (\mathsf{nonce}, e, \sigma)$
PartDec(params, $pk_{\mathsf{Sndr}} = \mathsf{ObfFunc}, \overrightarrow{pv} = \{pv_i\}_{i \in \mathcal{R}}, sv_i, c = (\mathsf{nonce}, e, \sigma)$):

    Let idx be the index of the public value corresponding to the secret value $sv_i$ in a
    lexicographic ordering of $\{pv_i\}_{i \in \mathcal{R}}$
    $d_i \leftarrow \mathsf{ObfFunc}(\overrightarrow{pv}, \mathsf{idx}, sv_i, \mathsf{nonce}, \sigma)$
    return $d_i$
FinalDec(params, $c = (\mathsf{nonce}, e, \sigma), \{d_i\}_{i \in \mathcal{R}' \subset \mathcal{R}}$):
    $w \leftarrow \mathsf{SS.Reconstruct}(\{d_i\}_{i \in \mathcal{R}' \subset \mathcal{R}})$
    $m \leftarrow e - w$
    return $m$
```

Construction 1: Obfuscation-Based ATE

3.1 *t*-Flexibility

For simplicity, we describe obfuscation-based ATE in a way that is not by default t-flexible, since the threshold t is fixed within the sender's public key. However, it can be made t-flexible in a very straightforward way, simply by including t as part of the (signed) input to the obfuscated program.

3.2 Reducing the Public Key Size

In the construction described above, the sender's public key size is polynomial in the number n of recipients. We can decrease the size of the public key by relying on differing-inputs obfuscation (diO) [BGI$^+$01,ABG$^+$13] instead of indistinguishability obfuscation (iO). If we do, then we can modify the obfuscated program to take a Merkle hash commitment to the set of recipients' public keys, instead of the entire list; additionally, we will be able to replace constrained signatures with any signature scheme. This will enable us to go from $poly(n)$-size public keys to $poly(t)$-size public keys. (We still need $poly(t)$ because that is the number of secret shares we must hard-code in the program in one of the hybrids in our security proof.)

4 Lower Bounds on Ciphertext Size for Recipient-Set-Oblivious Ad Hoc Threshold Encryption Schemes

One downside of the ATE scheme described in Section 3 is that it is not recipient-set-oblivious (Definition 1); that is, the decryption algorithms need the set of recipient public keys as input.

One might hope to be able to combine the sender-compactness of the ATE scheme in Section 3 with recipient-set-obliviousness. However, in this section, we show that any recipient-set-oblivious ad hoc threshold encryption scheme must have ciphertext sizes linear in the number of recipients. In Theorem 2, we consider the special case of broadcast encryption (that is, $t = 0$); then, in Theorem 3, we consider the general threshold encryption case.

Theorem 2. *In any recipient-set-oblivious ad hoc threshold encryption scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{FinalDec})$*, the average size of a ciphertext* c *produced as*

$$(\mathsf{params}) \leftarrow \mathsf{Setup}(1^\lambda, t = 0)$$
$$\{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\mathsf{params})\}_{i \in [u]}$$
$$\mathcal{R} \leftarrow \textit{ a random size-}n\textit{ subset of } [u]$$
$$c \leftarrow \mathsf{Enc}(\mathsf{params}, \{pk_i\}_{i \in \mathcal{R}}, m)$$

for any m *in the message space is* $O(\log_2 \binom{u}{n})$*.*

Proof. Imagine that Alice runs all four lines described above (that is, generates u key pairs, a random set $\mathcal{R}$ of recipients and a ciphertext). She then sends all the key pairs to Bob.

Later, she can use c to communicate to Bob her choice of $\mathcal{R}$. Bob attempts decryption with each key pair. For $t = 0$, one key pair is sufficient to decrypt; since our scheme is recipient-set-oblivious, Bob does not need to use the public keys belonging to other recipients. Thus, Bob identifies those for whom decryption yields m as belonging to $\mathcal{R}$. (Note that the probability of correct decryption with a key that does not belong to $\mathcal{R}$ should be negligible, or the threshold encryption scheme is not secure.)

Since there are $\binom{u}{n}$ possibilities for $\mathcal{R}$, Alice must use at least $\log_2 \binom{u}{n}$ bits to communicate $\mathcal{R}$ to Bob. Since the key pairs are generated independently of $\mathcal{R}$, they don't count towards those bits; thus, the ciphertext should be at least $\log_2 \binom{u}{n}$ bits long. In the case when $u = 2n$, this is lower-bounded by 2^n.

Theorem 3. *For any $t < n$, any recipient-set-oblivious ad hoc threshold encryption scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{FinalDec})$*, the average size of a ciphertext c produced as*

$$(\mathsf{params}) \leftarrow \mathsf{Setup}(1^\lambda, t)$$

$$\{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\mathsf{params})\}_{i\in[u]}$$

$$\mathcal{R} \leftarrow \text{ a random size-}n\text{ subset of } [u]$$

$$c \leftarrow \mathsf{Enc}(\mathsf{params}, \{pk_i\}_{i\in\mathcal{R}}, m)$$

for any m in the message space is $O(\log_2 \binom{u-t}{n-t})$.

Proof. The proof goes exactly as it does for Theorem 2, but Alice identifies to Bob t key pairs in $\mathcal{R}$, so that Bob only needs to identify the $n-t$ remaining ones. Bob does this by computing partial decryptions using the t identified key pairs, and testing each of the remaining $u-t$ key pairs. He tests a key pair by using it to generate a partial decryption and seeing whether that partial decryption combines with the ones it already has to produce m.

5 Recipient-Compact Homomorphic Ad Hoc Threshold Encryption

In this section, we describe three recipient-compact HATE constructions. In addition to recipient-compactness, all three of these schemes have *recipient-local evaluation*, meaning that each recipient can perform evaluation locally given just their compact component of the ciphertext.

Two of them (Section 5.1) are based on the share-and-encrypt paradigm. These are recipient-set-oblivious, but are not sender-compact. The last (Section 5.2) achieves sender-compactness by combining share-and-encrypt with the obfuscation-based sender-compact ATE from Section 3. However, like the ATE in Section 3, it is not recipient-set-oblivious.

5.1 Building HATE from Homomorphic Encryption and Secret Sharing

In this section, we describe our share-and-encrypt homomorphic ad hoc threshold encryption scheme which, despite its $\Theta(n)$-size ciphertexts, is efficient enough to be used in practice in some scenarios, because it is recipient-compact.

As we mentioned in the introduction, one natural way to build ATE is to use a threshold secret sharing scheme SS together with a public-key encryption scheme PKE. The idea is to secret share the message, and to encrypt each share to a different recipient using their public key; therefore, we call this the *share-and-encrypt* paradigm. We elaborate on it in Appendix D.

Notice that we are able to omit all but the relevant part of the ciphertext as input to $\mathsf{PartDec}$ for each party (where the relevant part is the one encrypted under their key), making the scheme both recipient-set-oblivious and recipient-compact. This further saves on communication in some contexts (Section 6.3).

Theorem 4. *Share-and-encrypt (described formally in Appendix D, Construction 3) is a (n, t)-statically secure (Definition 4, modifed to be static instead of super-static), recipient-set-oblivious, recipient-compact ATE, as long as* SS *is a secure share simulatable t-out-of-n secret sharing scheme, and* PKE *is a CPA-secure public key encryption scheme.*

We prove Theorem 4 in Appendix D.3.

If the secret sharing and encryption schemes are homomorphic in compatible ways, the share-and-encrypt construction is a Homomorphic ATE. The trick is finding the right homomorphic secret sharing and encryption schemes. In particular, if the secret sharing scheme is $\mathcal{F}$-homomorphic, the encryption scheme must be $\mathcal{F}'$-homomorphic, where $\mathcal{F}'$ includes the homomorphic evaluation of $\mathcal{F}$ over secret shares.

Of course, if the secret sharing and encryption schemes are both *fully* homomorphic, they give fully homomorphic ATE. However, no homomorphic threshold secret sharing schemes (with homomorphism over multiple inputs, without pre-distributed correlated randomness) is known, to the best of our knowledge.[4]

We show two efficient combinations of secret sharing and encryption which result in additively homomorphic ATE: Shamir-and-ElGamal (described in detail in Appendix D.4.1) and CRT-and-Paillier (Appendix D.4.2).

Shamir-and-ElGamal. We build share-and-encrypt HATE out of ElGamal encryption [ElG84] and a variant of Shamir secret sharing. We need to use a *variant* of Shamir secret sharing (which we call exponential Shamir secret sharing), and not Shamir secret sharing itself, because Shamir secret sharing is additively homomorphic (and the homomorphism is applied via addition of individual shares), so we would need the encryption scheme to support addition; however, ElGamal is only multiplicatively homomorphic, so if we attempt to apply a homomorphism on encrypted shares, it will not work. What we need in order to get an additively homomorphic ATE scheme is to use ElGamal encryption with a secret sharing scheme which is additively homomorphic, but whose homomorphism is applied via multiplication. Therefore, we need to alter our Shamir secret sharing scheme by moving the shares to the exponent; then, taking a product of two shares will result in a share of the sum of the two shared values. We refer to Appendix D.4.1 for a description of the ElGamal encryption scheme and the exponential Shamir secret sharing scheme which we use.

Theorem 5. *Shamir-and-ElGamal (described in Appendix D.4.1) is an additively homomorphic ad hoc threshold encryption scheme for a polynomial-size message space.*

Shamir-and-ElGamal is an ad hoc threshold encryption scheme by Theorem 4; the homomorphism follows from the homomorphisms of the underlying encryption and secret sharing schemes.

[4] Boyle *et al.* [BGI$^+$18] give a nice introduction to homomorphic secret sharing. Jain *et al.* [JRS17] and Dodis *et al.* [DHRW16] both build (threshold) function secret sharing, which gives homomorphic secret sharing, but the homomorphism is only over a single input.

In Shamir-and-ElGamal we are limited to polynomial-size message spaces since final decryption uses brute-force search to find a discrete log. Jumping ahead to LOVE MPC, polynomial-size message spaces are still useful in many applications, as explained in the introduction. Moreover, the server already does work that is polynomial in the number of users, so asking it to perform another polynomial computation is not unreasonable.

CRT-and-Paillier. We also build share-and-encrypt HATE out of Camenisch-Shoup encryption and Chinese Remainder Theorem based secret sharing. The Camenisch-Shoup encryption scheme is a variant of Paillier encryption that supports additive homomorphism. However, we cannot combine it with Shamir secret sharing, since Shamir shares all live in the same group, while each instance of a Camenisch-Shoup encryption scheme uses a different modulus. Therefore, we combine Camenisch-Shoup encryption with CRT secret sharing, which has exactly the property that different shares can live in different groups. Unlike Shamir-and-ElGamal (Appendix D.4.1), this HATE allows us to use large message spaces. We refer to Appendix D.4.2 for a description of the Camenisch-Shoup encryption scheme and the CRT secret sharing scheme which we use.

Theorem 6. *CRT-and-Paillier (described in Appendix D.4.2) is an additively homomorphic ad hoc threshold encryption scheme.*

CRT-and-Paillier is an ad hoc threshold encryption scheme by Theorem 4; the homomorphism follows from the homomorphisms of the underlying encryption and secret sharing schemes.

5.2 Building HATE from Obfuscation

As described in Section 3, the obfuscation-based ATE is not homomorphic. Informally, in order to make the obfuscation-based ATE $\mathcal{F}$-homomorphic, we can modify the obfuscated program to:

1. Use a $\mathcal{F}$-homomorphic secret sharing scheme [BGI16]. (As an example, Shamir secret sharing is additively-homomorphic.) Note that $\mathcal{F}$ should always include subtraction from a constant (in the appropriate group); the obfuscated program returns shares of the mask w, which we want to use, together with the masked message e, to obtain shares of $m = e - w$.
   However, this alone is not enough; even if the secret shares returned by the obfuscated programs are homomorphic, in order to extract them from the ciphertext, one must know a recipient secret value sv, while evaluation should require no secrets.
2. Use $\mathcal{F}'$-homomorphic public key encryption and decryption keys pk_i, sk_i instead of public and private values $pv_i = \mathsf{PRG}(sv_i), sv_i$. The obfuscated program would then not require sk_i as input; instead, it would return a ciphertext that requires sk_i for decryption.
   $\mathcal{F}'$ must include the functions necessary to evaluate $\mathcal{F}$ on the homomorphic secret shares.

This modification makes the construction $\mathcal{F}$-homomorphic while preserving sender-compactness. Thus, anyone (e.g., a server) can evaluate the obfuscated program to extract encryptions of all recipients' shares of the mask, homomorphically convert these into encrypted shares of the message, and homomorphically compute on those encrypted shares (since our public key encryption scheme is homomorphic, as are secret shares). The server would then send all parties their encrypted share of the computation output. Additionally, this construction is recipient-compact (as long as homomorphic shares are small), since each party only needs one compact part of the ciphertext for partial decryption.

More concretely, we can use ElGamal encryption [ElG84]. Once the obfuscated program is evaluated, we are essentially using the Shamir-and-ElGamal HATE (Section 5.1). In particular, this implies that we are limited to polynomial-size message spaces, since final decryption uses brute-force search to find a discrete logarithm.

In Appendix E we give more details about our homomorphic recipient-compact HATE construction. Construction 4 describes the construction; Algorithm 4 describes the new program that needs to be obfuscated and included in each sender's public key.

Theorem 7. *The modified obfuscation-based ATE (Construction 4, described in Appendix E) is (n, t)-super-statically secure (Definition 4) for any polynomial n, t, as long as* iO *is a secure indistinguishability obfuscator,* PPRF *is a secure puncturable* PRF*,* SS *is a secure secret sharing scheme,* SIG *is a constrained signature scheme, and* PKE *is a secure public-key encryption scheme. Moreover, it is $\mathcal{F}$-homomorphic if* SS *is $\mathcal{F}$ homomorphic (where $\mathcal{F}$ includes subtraction from a constant), and if* PKE *is $\mathcal{F}'$-homomorphic (where $\mathcal{F}'$ includes the evaluation of $\mathcal{F}$ on* SS *secret shares).*

6 Large-scale One-server Vanishing-participants Efficient MPC (LOVE MPC)

One attractive application of homomorphic ad hoc threshold encryption with sender-compactness is Large-scale One-server Vanishing-participants Efficient MPC (LOVE MPC). LOVE MPC is different from more traditional MPC in two ways: (1) in addition to tolerating corruptions, it tolerates some number of parties who vanish (i.e., drop out mid-computation), and (2) only the server learns the output. The vanishing participant property was introduced in the work of Badrinarayanan *et al.* [BJMS18], which called such parties "lazy parties".

6.1 Lower Bounds

We show lower bounds both for the number of message flows in a LOVE MPC construction, and for the setup requirements.

Theorem 8. *For many functions (including addition), a LOVE MPC cannot be instantiated in fewer than three message flows, and if only three flows are used, then setup (e.g. correlated randomness or PKI) is unavoidable.*

Proof. We prove this theorem in two parts.

Lower Bounds on Number of Message Flows. A one-message protocol (where each user sends the server a single message, as in non-interactive MPC (NIMPC) [BGI$^+$14a]) is impossible in our setting for many functions f, for the following reason. In a one-message protocol, the users would all send a single message to the server, who would compute the desired output. However, if the protocol is fault-tolerant, the set of participating users cannot be known in advance. Thus, an honest-but-curious server would be able to compute f on many different subsets of participating users, simply by ignoring some of the received messages. For example, if f is the sum of the users' individual values, the server could compute f both with and without a particular user present, thus learning every user's input.

A two-message protocol does not make sense, since a second message flow would involve the server sending the users messages. A server-to-user message before the user-to-server message does not solve the above problem, and a server-to-user message after the user-to-server message cannot affect the output, since the server should be the one to arrive at the output. We conclude that a LOVE MPC construction requires at least three message flows.

Lower Bounds on Setup Assumptions. A three-message protocol without any joint setup (e.g. correlated randomness or PKI) allows the server to perform what is essentially a Sybil attack. By fault-tolerance, the output should still be computable if a few participants drop out after sending the first message. Moreover, the output should not change depending on which participants drop out between the first and third flows; otherwise, the honest-but-curious server can pretend some users dropped out and see how the output changes (just like in the argument against one-message protocols). Therefore, the output should be fixed as soon as the second flow messages are sent by the server. (Generalizing to more than three message flows, the output should be fixed as soon as the server sends its last message.) This feature enables an honest-but-curious server to compute f on any single real user's input combined with inputs of the server's choice, as follows. After receiving the first message from a real user, the server will simulate the first message of $n-1$ users with inputs of the server's choice, and then simulate the rest of the messages of the protocol as if the real user dropped out before sending its third message. As long as the protocol can tolerate a single user dropping out, the server will be able to compute the desired output. We conclude that a three-message LOVE MPC construction requires some setup.

6.2 Definitions

Our LOVE MPC ideal functionality, described in Figure 3, is a variant of the trusted party functionality of Badrinarayanan *et al.* [BJMS18], modified to support only a single output party (the server Srvr) and to allow functions with more than a single bit of output.

Let $\mathcal{U}$ be the set of all parties, $\mathcal{P}$ be the set of parties who do not drop out by the end of the protocol execution, and $\mathcal{C}$ be the set of corrupt parties. For correctness, we require that $|\mathcal{P}| > t$ for a dropout threshold t. For security, we require that $|\mathcal{C}| \leq t_c$ for a corruption threshold t_c. Note that in all of our constructions, we have $t = t_c$.

A static semi-honest adversary $\mathcal{A}$ specifies the following sets: $\mathcal{C} \subseteq \mathcal{U}$ of corrupt parties such that $|\mathcal{C}| \leq t_c$, $\mathcal{D}_{\text{INPUT}} \subseteq \mathcal{U}$ of parties who drop out before the end of the input phase, and $\mathcal{D}_{\text{OUTPUT}} \subseteq \mathcal{U}$ of parties who drop out after the input phase (where $\mathcal{P} = \mathcal{U} \backslash (\mathcal{D}_{\text{INPUT}} \cup \mathcal{D}_{\text{OUTPUT}})$). Only parties who do not drop out before the end of the input phase (that is, $i \in \mathcal{U} \backslash \mathcal{D}_{\text{INPUT}}$) have their inputs included in the computation. The adversary receives the view of all the parties in $\mathcal{C}$.

Informally, a LOVE MPC protocol has static semi-honest security if for all input vectors $\{x_i\}_{i \in \mathcal{U}}$ and all efficient adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ who interacts with the ideal functionality in Figure 3 and can simulate the view of $\mathcal{A}$. Badrinarayanan *et al.* [BJMS18] also discuss security against malicious parties, which we do not address here.

We present our protocols in the PKI model. Because we consider only honest-but-curious attackers, the PKI model does not require any additional trust: the clients could simply exchange public keys via the server in two additional message flows before the start of the protocol. The importance of the PKI model for our protocols is that this exchange is independent of the inputs and needs to happen only once; after that, the protocols can be run repeatedly with the same public keys.

There are multiple ways to model PKI formally: "global" setup (e.g., Canetti and Rabin [CR03], Canetti *et al.* [CDPW07] and Dodis *et al.* [DKSW09]), which uses key registration that is shared by multiple, possibly different, protocols; or "local" setup (e.g., Barak *et al.* [BCNP04]), in which key registration is per protocol instance. In any of these, since our adversary is semi-honest, the simulator is allowed to know the secret keys of the corrupted parties; in addition, local setup means that the security definition is weaker and the simulator is more powerful, because the simulator can simulate the setup and thus is able to know (or even decide) secret keys for the honest parties. The LOVE MPC protocol that we describe in Section 6.3 can be proven secure with global setup, unless it is instantiated with the keyed-sender obfuscation-based HATE (Section 5.2), in which case it requires local setup (but still can be run multiple times with the same PKI).

6.3 Three-Message LOVE MPC from HATE

Let $\mathsf{HATE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{FinalDec}, \mathsf{Eval})$ be a homomorphic ad hoc threshold encryption scheme. Assume the HATE has been set up (and so params is publicly available, and contains t), and that each party has already run KeyGen and that everyone's public keys have been distributed through a public key infrastructure. We describe a three-message HATE-based LOVE MPC in Construction 2.

```
Functionality $\mathcal{F}_f$, interacting with server Srvr and parties $P_i$, $i \in \mathcal{U}$.

INIT: On input (INIT) from the simulator $\mathcal{S}$:
    1. Initialize an empty map INPUTS from parties to their inputs.
INPUTABORT: On input (INPUTABORT, $\mathcal{D}_{\text{INPUT}}$) from the simulator $\mathcal{S}$, store $\mathcal{D}_{\text{INPUT}}$.
INPUT: On input (INPUT, $x_i$) from party $P_i$: Store INPUTS$[i] = x_i$.
OUTPUTABORT: On input (OUTPUTABORT, $\mathcal{D}_{\text{OUTPUT}}$) from the simulator $\mathcal{S}$, store $\mathcal{D}_{\text{OUTPUT}}$.
OUTPUT: On input (OUTPUT) from the simulator $\mathcal{S}$:
    1. Remove $i$ from INPUTS for $i \in \mathcal{D}_{\text{INPUT}}$.
    2. If $|\mathcal{U} \backslash (\mathcal{D}_{\text{INPUT}} \cup \mathcal{D}_{\text{OUTPUT}})| > t$: compute $y = f(\text{INPUTS})$.
    3. Else: set $y = \perp$.
    4. Output $y$ to the server Srvr.
```

Fig. 3: Ideal Functionality $\mathcal{F}_f$ for LOVE MPC Secure Against Semi-Honest Adversaries.

```
Flow 1: Each party $P_i$ sends a message to the server Srvr
    Each party $P_i$, $i \in \mathcal{U}$ does the following:
    1. Computes
        $c_i \leftarrow \mathsf{HATE.Enc}(\mathsf{params}, sk_{\mathsf{Sndr},i}, \{pk_j\}_{j \in \mathcal{U}}, x_i)$.
    2. Sends $c_i$ to Srvr.
Flow 2: Server Srvr sends a message to each party $P_i$
    Let $\mathcal{D}_{\text{INPUT}} \subseteq \mathcal{U}$ be the set of parties from whom the server Srvr did not receive a ciphertext.
    Srvr computes the sum ciphertext
        $c \leftarrow \mathsf{HATE.Eval}(\mathsf{params}, \{pk_i\}_{i \in \mathcal{U}}, \{c_i\}_{i \in \mathcal{U} \backslash \mathcal{D}_{\text{INPUT}}}, f)$
    and sends $c$ to all parties $i \in \mathcal{U} \backslash \mathcal{D}_{\text{INPUT}}$.
Flow 3: Each party $P_i$ sends a message to the server Srvr
    Each party $P_i$, $i \in \mathcal{U} \backslash \mathcal{D}_{\text{INPUT}}$ does the following:
    1. Computes
        $d_i \leftarrow \mathsf{HATE.PartDec}(\mathsf{params}, \{pk_j\}_{j \in \mathcal{U}}, sk_i, c)$.
    2. Sends $d_i$ to Srvr.
The server Srvr computes the output
    Let $\mathcal{D}_{\text{OUTPUT}} \subseteq \mathcal{U} \backslash \mathcal{D}_{\text{INPUT}}$ be the set of parties from whom the server Srvr got a ciphertext $c_i$,
    but not a partial decryption $d_i$. As long as $|\mathcal{P} = \mathcal{U} \backslash (\mathcal{D}_{\text{INPUT}} \cup \mathcal{D}_{\text{OUTPUT}})| > t$, Srvr computes
        $y \leftarrow \mathsf{HATE.FinalDec}(\mathsf{params}, \{pk_i\}_{i \in \mathcal{U}}, c, \{d_i\}_{i \in \mathcal{P}})$.
```

Construction 2: LOVE MPC for Function f From HATE in Three Rounds

Theorem 9. *HATE-based LOVE MPC (Construction 2) in the global-setup PKI model returns the correct output of f if fewer than t parties drop out. It is secure against t static semi-honest corruptions as long as* HATE *is a (n, t)-super statically secure (Definition 4) $\mathcal{F}$-homomorphic ATE construction such that $f \in \mathcal{F}$.*

Proof. Correctness is true by the correctness of the underlying HATE.

To prove security, we describe a simulator $\mathcal{S}$ in Figure 4. Since we require global setup, $\mathcal{S}$ does not have access to honest parties' secret keys. However, because of the honest-but-curious assumption, $\mathcal{S}$ does see corrupt parties' secret keys and randomness. $\mathcal{S}$ can simulate by encrypting 0 for each honest party in Flow 1 (without knowing their secret keys), and simulating the partial decryptions for each honest party in Flow 3 (again without knowing their secret keys),

1. The simulator $\mathcal{S}$ sends (Init) to the ideal functionality.
2. $\mathcal{S}$ runs the adversary $\mathcal{A}$ to determine $\mathcal{C}$, $\mathcal{D}_{\text{Input}}$, $\mathcal{D}_{\text{Output}}$, and sends those to the ideal functionality.
3. [Flow 1] For each honest party P_i, $i \in \mathcal{U} \backslash \mathcal{C}$, $\mathcal{S}$ encrypts 0 in place of the actual input:
$$c_i \leftarrow \mathsf{HATE.Enc}(\mathsf{params}, \{pk_j\}_{j \in \mathcal{U}}, 0).$$
4. [Flow 2] Whether the server Srvr is honest or semi-honest (that is, whether or not its role is played by the simulator), it will correctly compute the sum ciphertext c and send it to all parties.
5. [Flow 3] For each corrupt party P_i, $i \in \mathcal{C}$, since the simulator $\mathcal{S}$ knows sk_i and the randomness used by the adversary, $\mathcal{S}$ can compute the partial decryption $d_i \leftarrow \mathsf{HATE.PartDec}(\mathsf{params}, \{pk_j\}_{j \in \mathcal{U}}, sk_i, c)$, which is guaranteed to equal the one the corrupt party will send the server Srvr in the final flow. ($\mathcal{S}$ can do this even if the adversary is rushing.) However, since $\mathcal{S}$ does not know the honest parties' keys, $\mathcal{S}$ must simulate their partial decryptions. $\mathcal{S}$ sends (Output) to the ideal functionality (note that the output is fixed at this point, since it is fixed as soon as the server sent the ciphertext), learns the actual output y, and computes the simulated partial decryptions for the honest parties by using the partial decryption simulatability (Definition 3) of the HATE. That is, the simulator runs
$$\{d_i\}_{i \in \mathcal{U} \backslash \mathcal{C}} \leftarrow \mathsf{HATE.SimPartDec}(\mathsf{params}, \{pk_i\}_{i \in \mathcal{U}}, c, \{d_i\}_{i \in \mathcal{C}}, y).$$
By partial decryption simulatability, these will be indistinguishable from genuine partial decryptions.
6. [Output Computation] Whether the server Srvr is honest or semi-honest, it will correctly compute the output from the partial decryptions it receives.

Fig. 4: Simulator $\mathcal{S}$ for LOVE MPC from HATE

by first performing partial decryption on behalf of the corrupt parties using their secret keys and randomness.

Notice that the only points in which the simulation differs from a real execution view is Flow 1, when the simulator encrypts 0s instead of the actual inputs, and Flow 3, when the simulator simulates partial decryptions instead of using genuine ones. The simulated corrupt parties' view is indistinguishable from a real view by CPA security and partial decryption simulatability, respectively.

Efficiency. Shamir-and-ElGamal LOVE MPC and CRT-and-Paillier LOVE MPC require $\Theta(n)$ communication per party, where $n = |\mathcal{U}|$. Since ciphertexts are $\Theta(n)$ in size, each party sends a $\Theta(n)$-size message in Flow 1, and receives a $\Theta(n)$-size message in Flow 2. However, we can leverage the recipient-compactness of share-and-encrypt and save some concrete cost by having the server only send each party the relevant part of the ciphertext in Flow 2; that is, the encryption of their secret share.

The advantage of obfuscation-based LOVE MPC is that it requires only constant communication per party once public keys have been distributed. Additionally, each party's public key need only be communicated to the server (perhaps with a PKI as an intermediary) but not to the other parties, since the server is the only one who needs to evaluate each party's obfuscated program. (Parties do need their peers' encryption keys in order to compute their ciphertexts for Flow 1, but those encryption keys comprise a small, constant-size component of each public key.)

Acknowledgements

We would like to thank Ran Canetti and Ben Kreuter for helpful discussions.

References

AB06. C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Trans. Inf. Theor.*, 29(2):208–210, September 2006.

ABG+13. Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. `http://eprint.iacr.org/2013/689`.

BBG+20. James Bell, K. A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. Cryptology ePrint Archive, Report 2020/704, 2020. `https://eprint.iacr.org/2020/704`.

BCNP04. Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th FOCS*, pages 186–195. IEEE Computer Society Press, October 2004.

BGG+18. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.

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

BGI+14a. Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 387–404. Springer, Heidelberg, August 2014.

BGI14b. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016.

BGI+18. Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018*, volume 94, pages 21:1–21:21. LIPIcs, January 2018.

BIK+17. Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1175–1191. ACM Press, October / November 2017.

BJMS18. Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure MPC: Laziness leads to GOD. Cryptology ePrint Archive, Report 2018/580, 2018. `https://eprint.iacr.org/2018/580`.

BM82. Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd FOCS*, pages 112–117. IEEE Computer Society Press, November 1982.

BW13. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

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

CDPW07. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Heidelberg, February 2007.

CFY17. Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. Catching MPC cheaters: Identification and openability. In Junji Shikata, editor, *ICITS 17*, volume 10681 of *LNCS*, pages 110–134. Springer, Heidelberg, November / December 2017.

CR03. Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, Heidelberg, August 2003.

CS03. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, August 2003.

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

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

DHMR08. Vanesa Daza, Javier Herranz, Paz Morillo, and Carla Ràfols. Ad-hoc threshold broadcast encryption with shorter ciphertexts. *Electr. Notes Theor. Comput. Sci.*, 192(2):3–15, 2008.

DHRW16. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

DKSW09. Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 146–162. Springer, Heidelberg, March 2009.

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

ElG84. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

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

GMR88. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

Hir10. Martin Hirt. Receipt-free K-out-of-L voting based on elgamal encryption. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000 of *Lecture Notes in Computer Science*, pages 64–82. Springer, 2010.

HS00. Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 539–556. Springer, Heidelberg, May 2000.

JRS17. Aayush Jain, Peter M. R. Rasmussen, and Amit Sahai. Threshold fully homomorphic encryption. Cryptology ePrint Archive, Report 2017/257, 2017. `http://eprint.iacr.org/2017/257`.

KPTZ13. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

MW16. Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

Row05. J.K. Rowling. *Harry Potter and the Half-Blood Prince*. Bloomsbury, 2005.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

SW14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

Zha16. Mark Zhandry. How to avoid obfuscation using witness PRFs. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 421–448. Springer, Heidelberg, January 2016.

A Background for Indistinguishability Obfuscation-Based Constructions

In our obfuscation-based constructions, we leverage indistinguishability obfuscation [BGI$^+$01], puncturable pseudorandom functions (PPRFs) [KPTZ13,BW13,BGI14b,SW14], secret sharing [Sha79], and constrained signatures [BZ14], all of which we describe below. Finally, we use pseudorandom generators [BM82], for which we do not provide a formal description since it is such a standard primitive.

A.1 Indistinguishability Obfuscation

Informally, indistinguishability obfuscation is a way to obfuscate a program in such a way that no efficient adversary can distinguish between the obfuscations of two programs of the same size as long as their input-output behaviors are the same. Indistinguishability obfuscation was first defined by Barak *et al.* [BGI$^+$01], and a candidate construction was proposed by Garg *et al.* [GGH$^+$13].

We restate the indistinguishability obfuscation definition of Garg *et al.* below (previously restated and rephrased by Boneh and Zhandry [BZ14]) with the simplification that we fix the parameter l to be circuit size.

Definition 6 (Definition 1 from Garg *et al.* [GGH$^+$13] / Definition 2.1 from Boneh and Zhandry [BZ14]). *An* indistinguiability obfuscator iO *for circuits is an efficient algorithm satisfying the following conditions:*

- $\mathsf{iO}(C)$ *preserves the functionality of the circuit C. That is, for all circuits C, for all inputs x, we have that*

$$\Pr[\mathsf{ObfC} \leftarrow \mathsf{iO}(C) : \mathsf{ObfC}(x) = C(x)] = 1.$$

- *For any two circuits C_0, C_1 of the same size l with the same functionality, the circuits $\mathsf{ObfC}_0 = \mathsf{iO}(C_0)$ and $\mathsf{ObfC}_1 = \mathsf{iO}(C_1)$ are indistinguishable. That is, for any efficient distinguisher D, there exists a negligible function negl such that the following holds: For all circuit sizes $l \in \mathbb{N}$, for all pairs of circuits C_0, C_1 of size l, we have that if $C_0(x) = C_1(x)$ for all inputs x, then*

$$|\Pr[\mathsf{D}(\mathsf{iO}(C_0)) = 1] - \Pr[\mathsf{D}(\mathsf{iO}(C_1)) = 1]| \leq \frac{1}{2} + negl(l).$$

A.2 Puncturable Pseudorandom Functions

A puncturable pseudorandom function (PPRF) [KPTZ13,BW13,BGI14b,SW14] is a pseudorandom function (PRF) whose keys can be punctured. Let $k\{x\}$ denote the PPRF key k punctured at point x; then $\mathsf{PPRF}_k(x') = \mathsf{PPRF}_{k\{x\}}(x')$ for all $x' \neq x$, but given $k\{x\}$, $\mathsf{PPRF}_k(x)$ is indistinguishable from random.

We give a more formal definition of puncturable pseudorandom functions below.

Definition 7. *A* puncturable pseudorandom function PPRF *consists of three algorithms:* KeyGen*,* Eval *and* $\mathsf{Puncture}$*.*

$\mathsf{KeyGen}(1^\lambda) \rightarrow k$ *sets up the* PPRF *secret key k.*
$\mathsf{Eval}(k, x) \rightarrow y$ *evaluates the* PPRF *at point* $x \in \mathsf{domain}$ *to obtain an output* $y \in \mathsf{range}$ *for polynomial-size sets* domain *and* range*. In the rest of this paper, we use the alternative notation* $\mathsf{PPRF}_k(x)$ *to denote* $\mathsf{Eval}(k, x)$*.*
$\mathsf{Puncture}(k, x) \rightarrow k\{x\}$ *outputs a punctured* PPRF *key* $k\{x\}$*.*

Furthermore, the algorithms must satisfy the following conditions.

- $(\mathsf{KeyGen}, \mathsf{Eval})$ *must be a secure PRF.*
- *Functionality should be preserved over all unpunctured inputs. That is, for all inputs* $x^* \in \mathsf{domain}$ *and keys k, if* $k\{x^*\} \leftarrow \mathsf{Puncture}(k, x^*)$*, then for all inputs* $x \neq x^*$*,* $\mathsf{PPRF}_k(x) = \mathsf{PPRF}_{k\{x^*\}}(x)$*.*
- *The true value of the* PPRF *at the punctured point x^* is indistinguishable from random given just $k\{x^*\}$.*

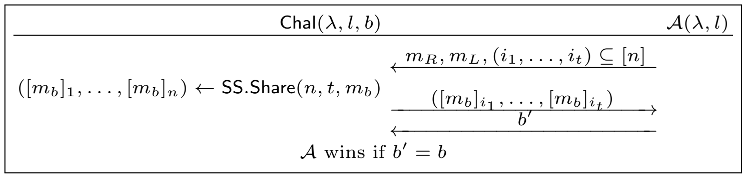
$\mathsf{Chal}(\lambda, l, b)$
$\mathcal{A}(\lambda, l)$
$m_R, m_L, (i_1, \ldots, i_t) \subseteq [n]$
$([m_b]_1, \ldots, [m_b]_n) \leftarrow \mathsf{SS.Share}(n, t, m_b)$
$([m_b]_{i_1}, \ldots, [m_b]_{i_t})$
b'
$\mathcal{A}$ wins if $b' = b$

Fig. 5: Privacy Game for Secret Sharing

A.3 Secret Sharing

Secret sharing was introduced by Shamir [Sha79]. Informally, a t-out-of-n threshold secret sharing of a secret m is an encoding of the secret into n pieces, or *shares*, such that any $t+1$ shares together can be used to reconstruct the secret m, but t or fewer shares give no information at all about m.

Definition 8. *A secret sharing scheme* SS *consists of two algorithms:* $\mathsf{SS.Share}$ *and* $\mathsf{SS.Reconstruct}$.

$\mathsf{SS.Share}(n, t, m) \to ([m]_1, \ldots, [m]_n)$ *takes in a secret m and produces the n secret shares (where $[m]_i$ denotes the ith share of m).*

$\mathsf{SS.Reconstruct}([m]_{i_1}, \ldots, [m]_{i_{t+1}}) \to \tilde{m}$ *takes in $t+1$ secret shares and returns the reconstructed secret $\tilde{m}$.*

Furthermore, the algorithms must satisfy the following two conditions.

Correctness *For all inputs m, for all polynomial $n, t < n$, for all sets of indices $[i_1, \ldots, i_{t+1}] \subset [n]$,*

$$\Pr\left[\begin{matrix}([m]_1, \ldots, [m]_n) \leftarrow \mathsf{SS.Share}(n, t, m), \\ \tilde{m} \leftarrow \mathsf{SS.Reconstruct}([m]_{i_1}, \ldots, [m]_{i_{t+1}})\end{matrix} : \tilde{m} = m\right] = 1$$

Privacy *Informally, privacy requires that given t or fewer shares of either m_R or m_L, no efficient adversary can guess which message was shared.*
For $b \in \{R, L\}$, let $\mathsf{EXP}(\mathcal{A}, \lambda, n, t, b)$ denote the game described in Figure 5 played with the adversary $\mathcal{A}$, security parameter λ, number of shares n, threshold t and fixed b. Let $\mathsf{WinProb}(\mathcal{A}, \lambda, n, t, b)$ denote the probability that the adversary $\mathcal{A}$ wins $\mathsf{EXP}(\mathcal{A}, \lambda, n, t, b)$. Then there exists a negligible function negl such that for all n and $t < n$ polynomial in λ and efficient adversaries $\mathcal{A}$,

$$|\mathsf{WinProb}(\mathcal{A}, \lambda, n, t, R) - \mathsf{WinProb}(\mathcal{A}, \lambda, n, t, L)| \leq \frac{1}{2} + negl(\lambda).$$

A.4 Constrained Signatures

Constrained signatures, introduced by Boneh and Zhandry [BZ14], are a special type of signature that supports constrained verification keys. Constrained verification keys reject all signatures on messages not matching some constraint C. Informally, constrained verification keys should be indistinguishable from real verification keys as long as no messages not matching the constraint have been signed.

We give a more formal definition of constrained signatures below.

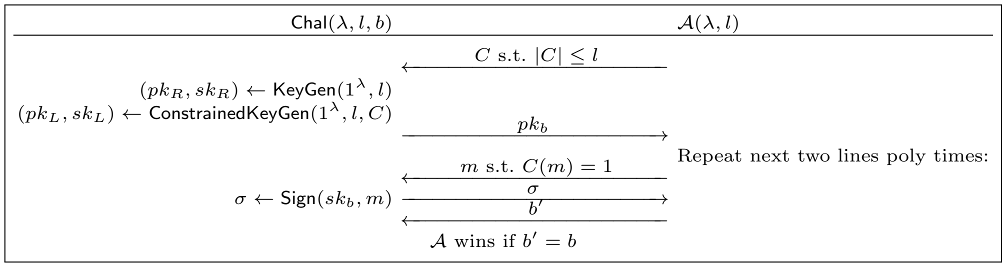
$\mathsf{Chal}(\lambda, l, b)$
$\mathcal{A}(\lambda, l)$
C s.t. $|C| \leq l$
$(pk_R, sk_R) \leftarrow \mathsf{KeyGen}(1^\lambda, l)$
$(pk_L, sk_L) \leftarrow \mathsf{ConstrainedKeyGen}(1^\lambda, l, C)$
pk_b
Repeat next two lines poly times:
m s.t. $C(m) = 1$
$\sigma \leftarrow \mathsf{Sign}(sk_b, m)$
σ
b'
$\mathcal{A}$ wins if $b' = b$

Fig. 6: Security Game for Constrained Signatures

Definition 9 (Definition 4.5 of Boneh and Zhandry [BZ14]). *A* constrained signature scheme SIG *consists of four algorithms:* KeyGen*,* Sign*,* Verify *and* $\mathsf{ConstrainedKeyGen}$*.*

$\mathsf{KeyGen}(1^\lambda, l) \to (pk, sk)$ *takes in the security parameter λ and an upper bound l on the constraint circuit. It outputs a valid verification/signing key pair (pk, sk). (We will often omit the parameter l.)*

$\mathsf{Sign}(sk, m) \to \sigma$ *signs the message m with the signing key sk.*

$\mathsf{Verify}(pk, m, \sigma) \to b$ *verifies the signature σ on the message m with the verification key pk. It returns 1 if the signature verifies, and 0 otherwise.*

$\mathsf{ConstrainedKeyGen}(1^\lambda, l, C) \to (pk, sk)$ *takes in the security parameter λ, an upper bound l on the constraint circuit, and a constraint circuit C such that $|C| \leq l$. It outputs a verification/signing key pair (pk_C, sk_C) such that* $\mathsf{Sign}(sk_C, m)$ *produces a valid signature relative to pk_C for all m such that $C(m) = 1$, but for any m where $C(m) = 0$, no valid signatures exist — that is,* $\mathsf{Verify}(pk_C, m, \sigma)$ *rejects all σ.*

Furthermore, the algorithms must satisfy the following two conditions.

- $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *must be a secure signature scheme (in the usual sense, e.g. existentially unforgeable [GMR88]).*
- *For $b \in \{R, L\}$, let* $\mathsf{EXP}(\mathcal{A}, \lambda, l, b)$ *denote the game described in Figure 6 played with the adversary $\mathcal{A}$, security parameter λ, constraint size upper bound l and fixed b. Let* $\mathsf{WinProb}(\mathcal{A}, \lambda, l, b)$ *denote the probability that the adversary $\mathcal{A}$ wins* $\mathsf{EXP}(\mathcal{A}, \lambda, l, b)$*. Then there exists a negligible function negl such that for all l polynomial in λ and efficient adversaries $\mathcal{A}$,*

$$|\mathsf{WinProb}(\mathcal{A}, \lambda, l, R) - \mathsf{WinProb}(\mathcal{A}, \lambda, l, L)| \leq \frac{1}{2} + negl(\lambda).$$

B Threshold Encryption Scheme: Threshold ElGamal

One simple example of a (non ad hoc) threshold encryption scheme is the threshold ElGamal scheme TEG due to Desmedt and Frankel [DF90], described in Figure 7. TEG is defined over a group $\mathcal{G}$ of prime order p with generator g in which the decisional Diffie-Hellman problem is assumed to be hard.

```
Setup$(1^\lambda, t)$:
    – Pick a secret key $sk \xleftarrow{\$} [p]$, and a random polynomial $f$ of degree $t$ with
      $sk$ as its $y$-intercept.
    – Return $pk = g^{sk}, \mathsf{msk} = f$.
KeyGen$(\mathsf{msk})$:
    – Pick a random $i \xleftarrow{\$} [1, \ldots, p-1]$.
    – Return $sk_i = f(i)$.
Enc$(pk, m \in \mathcal{G})$:
    – Pick a random $y \in [p]$.
    – $u = g^y$.
    – $v = (pk)^y m$.
    – Return $c = (u, v)$.
PartDec$(sk_j, c = (u, v))$:
    – Return $d_j = u^{sk_j}$.
FinalDec$(\{d_i\}_{i \in \mathcal{R}' \subseteq \mathcal{R}}, c = (u, v))$:
    – Interpolate the partial decryptions in the exponent to get $u^{sk}$: $y =
      \prod_{i \in \mathcal{R}' \subseteq \mathcal{R}} d_j^{\lambda_i}$, where $\lambda_i$ is the appropriate Lagrange coefficient. (The La-
      grange coefficients used depend on the identities $i$ of the parties who partic-
      ipate. However, given that a certain threshold of parties do, the Lagrange
      coefficients do not affect the output.)
    – Return $m = \frac{v}{y}$.
Eval$(pk, c_1 = (u_1, v_1), c_2 = (u_2, c_2), \times)$:
    – $u = u_1 u_2$
    – $v = v_1 v_2$
    – Return $c = (u, v)$.
```

Fig. 7: Threshold ElGamal Multiplicatively Homomorphic Encryption Scheme (TEG)

Lemma 1. *Threshold ElGamal is (n, t)-statically secure (Definition 4, modified to be static instead of super-static, and to use pk instead of $\{pk_i\}_{i\in\mathcal{U}}$) for any polynomial n, t as long as the Decisional Diffie-Hellman (DDH) assumption holds in $\mathcal{G}$.*

Informally, this lemma follows by a standard reduction from the DDH assumption.

Lemma 2. *Threshold ElGamal is (n, t)-partial decryption simulatable (Definition 3, modified to be static instead of super-static, and to use pk instead of $\{pk_i\}_{i\in\mathcal{U}}$) as long as the Decisional Diffie-Hellman assumption holds in $\mathcal{G}$.*

Informally, this lemma follows since partial decryptions can easily be simulated by interpolation in the exponent.

C Security of the Obfuscation-Based Ad Hoc Threshold Encryption Construction

Theorem 10 (Restated from Theorem 1). *The obfuscation-based ATE (Construction 1) is (n, t)-super-statically secure (Definition 4) for any polynomial n, t, as long as* iO *is a secure indistinguishability obfuscator,* PPRF *is a secure puncturable* PRF *with range $\mathbb{Z}_p$,* SS *is a secure secret sharing scheme scheme,* SIG *is a constrained signature scheme, and* PRG *is a secure pseudorandom generator with domain $\{0,1\}^{\lambda}$ and range in $\{0,1\}^{2\lambda}$.*

In order to prove Theorem 1, we must show the obfuscation-based Homomorphic Ad Hoc Threshold Encryption construction is super-statically semantically secure and super-statically partial decryption simulatable. We prove super-static semantic security in Appendix C.1; we prove partial decryption simulatability in Appendix C.2.

C.1 Proof that Obfuscation-Based Homomorphic Ad Hoc Threshold Encryption Share-and-Encrypt is Super-Statically Semantically Secure

Notation. In our sequence of games, we use $c^* = (\mathsf{nonce}^*, e^*, \sigma^*)$ to denote the challenge ciphertext. In particular, nonce^* denotes the value to which the PPRF is applied in order to generate the mask w^*, and e^* denotes the "one time pad" encryption of the challenge message with the generated mask ($e^* = m^* + w^*$ in the group $\mathcal{G}$).

We use $\overrightarrow{pv}$ to denote a vector of public values. $\overrightarrow{pv}$ is always assumed to be ordered lexicographically (that is, all programs implicitly reject vectors of public values which are out of order).

Proof. We show a sequence of indistinguishable games between a super-static semantic security challenger Chal and an adversary $\mathcal{A}$. The sequence starts with $\mathsf{EXP}(\mathcal{A}, \lambda, n, t, R)$ from Definition 2 (that is, a challenger who always uses $b = R$), and ends with $\mathsf{EXP}(\mathcal{A}, \lambda, n, t, L)$ (that is, a challenger who always uses $b = L$). We summarize this sequence of games in Figure 8. Instead of showing all of the games, we show that the first game is indistinguishable from a game where the challenge ciphertext encrypts 0; to get to the last game, the shown sequence of games is reversed with m_L instead of m_R.

For the sake of simplicity, we assume that the number of corrupt challenge ciphertext recipients $|\mathcal{R}^* \cap \mathcal{C}|$ is always t; security in this case trivially implies security for smaller $\mathcal{R}^* \cap \mathcal{C}$.

Game 1 This is $\mathsf{EXP}(\mathcal{A}, \lambda, n, t, R)$ as described in Definition 2 and Figure 1.

Game 2 In this game, when creating the sender's public key (specifically, the sender's signature verification key $\mathsf{SIG}.pk$), the challenger Chal generates a constrained verification key instead of a regular one. The constraint is designed to ensure that the challenge nonce nonce^* on which the PPRFs are called can only ever be associated with the challenge recipient set. That is, the challenger picks nonce^* at random when generating the sender public key, and sets the signature constraint to be

$$C(\overrightarrow{pv}, \mathsf{nonce}) = \begin{cases} 0, & \text{if } \mathsf{nonce} = \mathsf{nonce}^* \text{ and } \overrightarrow{pv} \neq \{pv_i\}_{i \in \mathcal{R}^*} \\ 1, & \text{otherwise} \end{cases}$$

When computing the challenge ciphertext c^*, Chal uses the nonce nonce^*.

Game 2 is indistinguishable from Game 1 by the security of constrained signatures. Since the challenger chooses each new nonce at random, and with overwhelming probability will never sign anything not satisfying the constraint (since no nonce other than the challenge nonce will be equal to nonce^*), then if the adversary can distinguish Game 2 from Game 1, then the challenger can use that adversary to distinguish between a constrained and unconstrained public verification key.

Game 3.k for $k \in [1, \ldots, n - t]$

In this game, when choosing the kth honest party's public value pv, the challenger chooses it at random from $\{0,1\}^{2\lambda}$, so that with overwhelming probability it has no preimages relative to the pseudorandom generator PRG.

Game 3.1 is indistinguishable from Game 2, and Game 3.k is indistinguishable from Game 3.$(k-1)$ for $k \in [2, \ldots, n-t]$, by the security of pseudorandom generators. We let Game 3 denote Game 3.$(n-t)$.

Game 4 In this game, when creating the sender's public key (specifically, the obfuscation of Algorithm 1 it contains), the challenger punctures the PPRF keys k_{Dec} and k_{Share}. To preserve the input-output behavior of the program, Chal hardwires $w^* = \mathsf{PPRF}_{k_{\mathsf{Dec}}}(\mathsf{nonce}^*)$ and $r^* = \mathsf{PPRF}_{k_{\mathsf{Share}}}(\mathsf{nonce}^*)$ into the program, in order to set $w = w^*$ and $r = r^*$ when $\mathsf{nonce} = \mathsf{nonce}^*$, as shown in Algorithm 2.

Algorithm 2 $f^{\mathsf{Game\ 4}}_{k_{\mathsf{Dec}}\{\mathsf{nonce}^*\}, k_{\mathsf{Share}}\{\mathsf{nonce}^*\}, \mathsf{SIG}.pk, \mathsf{nonce}^*, w^*, r^*}(\overrightarrow{pv}, \mathsf{idx}, sv, \mathsf{nonce}, \sigma)$

```
if ($\overrightarrow{pv}[\mathsf{idx}] = \mathsf{PRG}(sv)$) and ($\mathsf{SIG.Verify}(\mathsf{SIG}.pk, (\overrightarrow{pv}, \mathsf{nonce}), \sigma)$) then
    if $\mathsf{nonce} = \mathsf{nonce}^*$ then
        $w = w^*$
        $r = r^*$
    else
        $w \leftarrow \mathsf{PPRF}_{k_{\mathsf{Dec}}}(\mathsf{nonce})$
        $r \leftarrow \mathsf{PPRF}_{k_{\mathsf{Share}}}(\mathsf{nonce})$
    $[w]_{\mathsf{idx}} \leftarrow \mathsf{SS.Share}(w; r)[\mathsf{idx}]$ {This gives the idxth secret share of $w$}
    return $[w]_{\mathsf{idx}}$
```

Game 4 is indistinguishable from Game 3 by the security of indistinguishability obfuscation; the programs have identical input-output behavior.

Game 5 In this game, the challenger Chal chooses r^* truly at random.

Game 5 is indistinguishable from Game 4 by the security of puncturable PRFs.

Game 6 In this game, the challenger Chal chooses w^* truly at random.

Chal also computes the one time pad component of the challenge ciphertext e^* as $e^* = (m_R + w^*)$.

Game 6 is indistinguishable from Game 5 by the security of puncturable PRFs.

Game 7 In this game, the challenger Chal modifies the obfuscated program to hardcode the secret shares $[w^*]_{\mathsf{idx}}$ for indices idx corresponding to corrupt parties, as described in Algorithm 3. (Let $\mathcal{I}_{\mathcal{C}} \subseteq [n]$ denote the set of indices $\mathsf{idx} \in [n]$ corresponding to corrupt party indices in the lexicographic ordering of $\{pv_i\}_{i \in \mathcal{R}^*}$.) To preserve the input-output behavior of the program, Chal computes the shares $\{[w^*]_{\mathsf{idx}}\}_{\mathsf{idx} \in \mathcal{I}_{\mathcal{C}}}$ as $[w^*]_{\mathsf{idx}} \leftarrow \mathsf{SS.Share}(w^*; r^*)[\mathsf{idx}]$, exactly as they would have been computed in Algorithm 2.

Algorithm 3 $f^{\mathsf{Game\ 7}}_{k_{\mathsf{Dec}}\{\mathsf{nonce}^*\}, k_{\mathsf{Share}}\{\mathsf{nonce}^*\}, \mathsf{SIG}.pk, \mathsf{nonce}^*, \{[w^*]_{\mathsf{idx}}\}_{\mathsf{idx} \in \mathcal{I}_{\mathcal{C}}}}(\overrightarrow{pv}, \mathsf{idx}, sv, \mathsf{nonce}, \sigma)$

```
if ($\overrightarrow{pv}[\mathsf{idx}] = \mathsf{PRG}(sv)$) and ($\mathsf{SIG.Verify}(\mathsf{SIG}.pk, (\overrightarrow{pv}, \mathsf{nonce}), \sigma)$) then
  if $\mathsf{nonce} = \mathsf{nonce}^*$ then
    $[w]_{\mathsf{idx}} = [w^*]_{\mathsf{idx}}$
  else
    $w \leftarrow \mathsf{PPRF}_{k_{\mathsf{Dec}}}(\mathsf{nonce})$
    $r \leftarrow \mathsf{PPRF}_{k_{\mathsf{Share}}}(\mathsf{nonce})$
    $[w]_{\mathsf{idx}} \leftarrow \mathsf{SS.Share}(w; r)[\mathsf{idx}]$ {This gives the idxth secret share of $w$}
  return $[m]_{\mathsf{idx}}$
```

Game 7 is indistinguishable from Game 6 by the security of indistinguishability obfuscation. The programs have identical input-output behavior for the following reasons:

- On nonce $\mathsf{nonce} \neq \mathsf{nonce}^*$, the program behavior is unchanged.
- We guarantee that the program returns nothing in both games for nonce^* and $\overrightarrow{pv} \neq \{pv_i\}_{i \in \mathcal{R}^*}$ since no signatures exist on $(\overrightarrow{pv}, \mathsf{nonce}^*)$ for $\overrightarrow{pv} \neq \{pv_i\}_{i \in \mathcal{R}^*}$.
- We guarantee that the program returns nothing in both games on nonce^* and indices idx corresponding to honest pv_i, $i \in \mathcal{R}^* \backslash \mathcal{C}$ since no values sv exist such that $pv_i = \mathsf{PRG}(sv)$ for honest parties i.

Game 8 In this game, the challenger Chal replaces the hardcoded secret shares of w^* with hardcoded secret shares of 0. Game 8 is indistinguishable from Game 7 by the security of the secret sharing scheme SS. The hardcoded shares of w^* must be indistinguishable from the hardcoded shares of 0.

Game 9 In this game, the challenger switches to using $m^* = 0$. Game 9 is indistinguishable from Game 8 because the distributions do not change at all; e^* is still uniformly random, and the obfuscated program, which no longer contains any information about w^*, is unaffected.

The rest of the games undo what we did up until this point, replacing m_R with m_L.

Game	Justification	$\mathsf{SIG}.pk$	Honest pv_i	Obfuscated Program	c^*	m^*
1		real	real	real	real	m_R
2	Constrained Signatures	constrained to only verify on $(\overrightarrow{pk}, \mathsf{nonce}^*)$ when $\overrightarrow{pk} = \{pv_j\}_{j \in \mathcal{R}^*}$				
3	PRG		no matching secrets			
4	iO			puncture k_{Dec} and k_{Share} at nonce^*; hardcode correct values w^* and r^*		
5	PPRF			hardcode random r^*		
6	PPRF			hardcode random mask w^*	compute e^* using the random mask	
7	iO			hardcode shares of w^* instead of w^* and r^*		
8	SS			hardcode shares of 0 instead of shares of w^*		
9	identical distributions					0

Fig. 8: Summary of Hybrids in Proof of Theorem 1

C.2 Proof that Obfuscation-Based Ad Hoc Threshold Encryption Share-and-Encrypt is Super-Partial Decryption Simulatable

$\mathsf{SimPartDec}$ is simply the secret sharing $\mathsf{SimShares}$ algorithm. An adversary who distinguishes such simulated shares from real shares can be used to break the super-static semantic security of the scheme.

D HATE from Homomorphic Encryption and Secret Sharing

In this section, we detail our share-and-encrypt homomorphic ad hoc threshold encryption scheme which, despite its $\Theta(n)$-size ciphertexts, is efficient enough to be used in practice in some scenarios.

D.1 Background

We leverage homomorphic public key encryption schemes and secret sharing. We assume familiarity with public key encryption schemes. We briefly review secret sharing below. In particular, we use one non-standard property of secret sharing, which is *share simulatability.* Informally, a share simulatable t-out-of-n threshold secret sharing scheme ($\mathsf{SS.Share}, \mathsf{SS.Reconstruct}$) has a third algorithm $\mathsf{SS.SimShares}$ which takes in a message m_R and t or fewer honestly generated shares for a different message m_L, and

$$\begin{array}{rcl}
\mathsf{Chal}(n, t, \mathsf{SimShares}, b) & & \mathcal{A}(n, t) \\
\hline
 & \xleftarrow{\mathcal{C}, m_R, m_L} & \\
\{[m_R]_i\}_{i\in[n]} \leftarrow \mathsf{SS.Share}(n, t, m_R) & & \\
\{[m_L]_i\}_{i\in[n]} \leftarrow \mathsf{SS.Share}(n, t, m_L) & & \\
\{[m'_L]_i\}_{i\in[n]\setminus\mathcal{C}} \leftarrow \mathsf{SimShares}(n, t, \{[m_R]_i\}_{i\in\mathcal{C}}, m_L) & & \\
\text{If } b = 0\text{: set } x = \{[m_L]_i\}_{i\in[n]} & & \\
\text{If } b = 1\text{: set } x = \{[m_R]_i\}_{i\in\mathcal{C}} \cup \{[m'_L]_i\}_{i\in[n]\setminus\mathcal{C}} & \xrightarrow{x} & \\
 & \xleftarrow{b'} & \\
\hline
\multicolumn{3}{c}{\mathcal{A} \text{ wins if } b' = b,\ |m_R| = |m_L| \text{ and } |\mathcal{C}| \le t}
\end{array}$$

Fig. 9: Share Simulatability Game for Secret Sharing

generates the remaining shares such that all of the shares together are indistinguishable from an honestly generated sharing of m_R. $\mathsf{SS.SimShares}$ is only used in the proofs, not in the construction.

Secret sharing was introduced by Shamir [Sha79]. Informally, a t-out-of-n threshold secret sharing of a secret m is an encoding of the secret into n pieces, or *shares*, such that any $t + 1$ shares together can be used to reconstruct the secret m, but t or fewer shares give no information at all about m. A secret sharing scheme SS consists of two algorithms: $\mathsf{SS.Share}$ and $\mathsf{SS.Reconstruct}$.

– $\mathsf{SS.Share}(n, t, m) \to ([m]_1, \dots, [m]_n)$ takes in a secret m and produces the n secret shares.
– $\mathsf{SS.Reconstruct}([m]_{i_1}, \dots, [m]_{i_{t+1}}) \to \tilde{m}$ takes in $t+1$ secret shares and returns the reconstructed secret $\tilde{m}$.

Informally, correctness requires that $\tilde{m} = m$, and privacy requires that given t or fewer shares of either m_R or m_L, no efficient adversary can guess which message was shared.

D.1.1 Share Simulatability

We additionally use a property which we call *share simulatability*, which requires that given t or fewer honestly generated shares of m_R and given m_L, there exists an efficient algorithm $\mathsf{SS.SimShares}$ which generates the rest of the shares in such a way that the resulding sharing is indistinguishable from a fresh sharing of m_L.

Definition 10 (Secret Sharing: Share Simulatability).
For $b \in \{0, 1\}$, let $\mathsf{EXP}(\mathcal{A}, n, t, \mathsf{SimShares}, b)$ denote the game described in Figure 9 played with the adversary $\mathcal{A}$, number of shares n, threshold t, simulation algorithm $\mathsf{SimShares}$ *and fixed b. Let $\mathsf{WinProb}(\mathcal{A}, n, t, \mathsf{SimShares}, b)$ denote the probability that the adversary $\mathcal{A}$ wins $\mathsf{EXP}(\mathcal{A}, n, t, \mathsf{SimShares}, b)$.*

A secret sharing scheme scheme ($\mathsf{SS.Share}, \mathsf{SS.Reconstruct}$) *is* (n, t)-share simulatable *if there exists an efficient simulation algorithm* $\mathsf{SS.SimShares}$ *such that for all efficient adversaries $\mathcal{A}$, there exists a negligible function negl such that*

$$|\mathsf{WinProb}(\mathcal{A}, n, t, \mathsf{SS.SimShares}, 0) - \mathsf{WinProb}(\mathcal{A}, n, t, \mathsf{SS.SimShares}, 1)| \le \frac{1}{2} + negl(\lambda).$$

```
Setup($1^\lambda$):
    $\mathsf{params}_{\mathsf{PKE}} \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
    $\mathsf{params}_{\mathsf{SS}} \leftarrow \mathsf{SS.Setup}(1^\lambda)$.
    return $\mathsf{params} = (\mathsf{params}_{\mathsf{PKE}}, \mathsf{params}_{\mathsf{SS}})$.
KeyGen(params):
    return $(pk, sk) \leftarrow \mathsf{PKE.KeyGen}(\mathsf{params}_{\mathsf{PKE}})$.
Enc(params, $\{pk_i\}_{i\in\mathcal{R}}, t, m$):
    $\{[m]_i\}_{i\in\mathcal{R}} \leftarrow \mathsf{SS.Share}(|\mathcal{R}|, t, m)$.
    return $c \leftarrow \{\mathsf{PKE.Enc}(pk_i, [m]_i)\}_{i\in\mathcal{R}}$.
PartDec(params, $sk_i, c_i$):
    return $d_i \leftarrow \mathsf{PKE.Dec}(sk_i, c_i)$.
FinalDec($\mathsf{params} = 1^\lambda, \{d_i\}_{i\in\mathcal{R}'\subseteq\mathcal{R},|\mathcal{R}'|>t}$):
    return $m \leftarrow \mathsf{SS.Reconstruct}(\{d_i\}_{i\in\mathcal{R}'\subseteq\mathcal{R},|\mathcal{R}'|>t})$.
```

Construction 3: Share-and-Encrypt Ad Hoc Threshold Encryption

D.1.2 Shamir Secret Sharing [Sha79]

Shamir t-out-of-n secret sharing (Shamir) uses degree-(t) polynomials over some field. $\mathsf{Shamir.Share}(n, t, m)$ generates a random degree-(t) polynomial f with m as its y-intercept; each share $[m]_i$ is a point $(x_i, f(x_i))$ on the polynomial (with $x_i \neq 0$). Any $t + 1$ shares can be used to interpolate the polynomial, reconstructing m. Any t or fewer shares give no information about m.

Shamir secret sharing is share simulatable; any t or fewer points can be interpolated with $(0, m_L)$ (and optionally with some additional random points) to obtain a degree-t polynomial.

Additionally, Shamir secret sharing is linearly homomorphic: a shared value m can be multiplied by a constant, or added to another shared value m', by separately operating on the individual shares.

D.2 Building ATE from Homomorphic Encryption and Secret Sharing

Let ($\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec}$) be our public-key encryption scheme, and let ($\mathsf{SS.Share}, \mathsf{SS.Reconstruct}$) be our share simulatable threshold secret sharing scheme. We also allow algorithms PKE.Setup and SS.Setup, which handle global setup for the encryption and secret sharing scheme, respectively. The share-and-encrypt ATE scheme is formally defined in Construction 3.

In Appendix D.3, we prove the security of the share-and-encrypt ATE. In Appendix D.4, we describe two homomorphic instantiations of the share-and-encrypt ATE:

1. *Shamir-and-ElGamal* uses exponential Shamir secret sharing and the ElGamal public key encryption scheme, and
2. *CRT-and-Paillier* uses Chinese Remainder Theorem secret sharing and a variant of Paillier encryption.

D.3 Proofs of Properties of the Share-and-Encrypt Ad Hoc Threshold Encryption Construction

In this section, we prove Theorem 4.

Theorem 11 (Restated from Theorem 4). *The share-and-encrypt ATE (Construction 3) is (n,t)-statically secure (Definition 4, modified to be static instead of super-static), as long as* SS *is a secure share simulatable t-out-of-n secret sharing scheme, and* PKE *is a CPA-secure public key encryption scheme.*

In order to prove Theorem 4, in Appendix D.3.1 we show that the share-and-encrypt ATE is (n,t)-statically semantically secure, and in Appendix D.3.2 we show that it is (n,t)-partial decryption simulatable.

D.3.1 Proof that Share-and-Encrypt is Statically Semantically Secure

Proof. We show a sequence of indistinguishable games between a static security challenger Chal and an adversary $\mathcal{A}$. The sequence starts with $\mathsf{EXP}(\mathcal{A},\lambda,n,t,R)$ from Definition 2 (that is, a challenger who always uses $b=R$), and ends with $\mathsf{EXP}(\mathcal{A},\lambda,n,t,L)$ (that is, a challenger who always uses $b=L$).

Game 1 This is the game as described in Figure 1 with $b=R$ (that is, this is $\mathsf{EXP}(\mathcal{A},\lambda,n,t,R)$).

Game 2 This game is the same as the previous game, but when computing the challenge ciphertext c^*, Chal does the following:

- $\{[m_L]_i\}_{i\in\mathcal{R}} \leftarrow \mathsf{SS.Share}(n,t,m_L)$
- $\{[m_R]_i\}_{i\in\mathcal{R}\cap\mathcal{C}} \leftarrow \mathsf{SS.SimShares}(n,t,\{[m_L]_i\}_{i\in\mathcal{R}\setminus\mathcal{C}},m_L)$
- $c^* \leftarrow \{\mathsf{PKE.Enc}(pk_i,[m_L]_i)\}_{i\in\mathcal{R}\setminus\mathcal{C}} \cup \{\mathsf{PKE.Enc}(pk_i,[m_R]_i)\}_{i\in\mathcal{R}\cap\mathcal{C}}$

If $\mathcal{A}$ can tell the difference between this game and the previous game, then we can design another adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the share simulatability property of the secret sharing scheme SS (Definition 10). $\mathcal{B}$ forwards $(m_R,m_L,\mathcal{R}\cap\mathcal{C})$ to the share simulatability challenger (Figure 9). Upon receiving shares from the share simulatability challenger, $\mathcal{B}$ encrypts them and sends them to $\mathcal{A}$. If the share simulatability challenger flips $b=0$, $\mathcal{A}$'s view will be as in the previous game; if the share simulatability challenger flips $b=1$, $\mathcal{A}$'s view will be as in this game. $\mathcal{B}$ sends the share simulatability challenger $b''=0$ if $\mathcal{A}$ submits $b'=R$, and $b''=1$ if $\mathcal{B}$ submits $b'=L$.

Game 3.idx for idx $\in[1,\dots,n-t]$

This game is the same as the previous game, but for the idxth honest party (without loss of generality, let that be P_i with public key pk_i), the challenger encrypts $[m_L]_i$. Game 3.1 is indistinguishable from Game 2, and Game 3.idx is indistinguishable from Game 3.(idx $-$ 1) for idx $\in[2,\dots,n-t]$, by the CPA security of the public key encryption scheme PKE. If $\mathcal{A}$ can tell the difference between these games, then we can design another adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the CPA security of PKE. $\mathcal{B}$ honestly generates all of the PKE keys except for the idxth honest key pair. $\mathcal{B}$ talks to a CPA PKE challenger to get pk_i. It then does everything as before, except it sends $m_0=[m_R]_i$ and $m_1=[m_L]_i$ to the CPA challenger, and uses the challenge ciphertext it gets as part of c^*. Note that when the CPA challenger uses $b=0$ we are in the previous game, and when the CPA challenger uses $b=1$ we are in this game. $\mathcal{B}$ passes on the guess made by $\mathcal{A}$ to the CPA challenger.

Note that Game 3.$(n-t)$ is $\mathsf{EXP}(\mathcal{A},\lambda,n,t,L)$.

D.3.2 Proof that Share-and-Encrypt is Partial Decryption Simulatable

Proof. $\mathsf{SimPartDec}$ can simulate partial decryptions in the share-and-encrypt ad hoc threshold encryption scheme in Construction 3 simply by running $\{d_i\}_{i\in\mathcal{R}\setminus\mathcal{C}} \leftarrow \mathsf{SS.SimShares}(\{d_i\}_{i\in\mathcal{R}\cap\mathcal{C}}, m_R)$, and returning $\{d_i\}_{i\in\mathcal{R}\setminus\mathcal{C}}$.

Any adversary $\mathcal{A}$ who can win the static partial decryption simulatability game described in Figure 2 when played with $\mathsf{SimPartDec}$ with non-negligible probability can be used to break the share simulatability of SS and win the share simulatability game described in Figure 9.

D.4 Share-and-Encrypt HATE Instantiations

In this appendix, we instantiate the share-and-encrypt HATE (Construction 3) in two ways.

D.4.1 Shamir-and-ElGamal We use ElGamal multiplicatively homomorphic encryption together with a variant of Shamir secret sharing to instantiate Shamir-and-ElGamal. As we explain in Section 5.1, because ElGamal is multiplicatively homomorphic, we need to use a secret sharing scheme in which the additive homomorphism is applied via multiplication. Additive homomorphism in the Shamir secret sharing scheme is applied via addition; therefore, we use a modified version of Shamir secret sharing, which we call exponential Shamir. Exponential Shamir secret sharing is the same as Shamir secret sharing, but with the Shamir shares in the exponent. The resulting scheme is additively homomorphic, where the homomorphism is applied via multiplication, like we wanted. The downside is that reconstruction now involves taking a discrete logartihm, which we only know how to do using brute force, so we are limited to polynomial-size message spaces.

ElGamal Multiplicatively Homomorphic Encryption. Figure 10 describes the ElGamal multiplicatively homomorphic encryption scheme (EG). Note that we split the key generation algorithm into two algorithms: Setup and KeyGen. This is because when we use ElGamal as part of our HATE scheme, it is important that all parties share the same modulus and generator, so we factor out part of KeyGen into Setup, which will only be run once globally.

Exponential Shamir Secret Sharing. Figure 11 describes the exponential Shamir secret sharing scheme ($\mathsf{EShamir}$).

Notice that the reconstruction uses brute force search; this means that this secret sharing scheme can only be used for very small (polynomial-size in λ) message spaces. However, HATE is interesting even in this setting. For instance, if all we want to do is take a poll by summing encryptions of 0s and 1s, this HATE scheme enables us to do it. It is reasonable to assume that the server can manage to do brute force search over a polynomial space, since it is already doing quadratic work in this computation.

Lemma 3. *The exponential Shamir secret sharing scheme (*$\mathsf{EShamir}$*) described in Figure 11 is share simulatable.*

Proof. Informally, given a message m and t or fewer shares, we obtain correctly distributed remaining shares by interpolating the given with $(0, g^m)$ (and possibly with random values, if fewer than t shares are provided) in the exponent. This is done in a manner similar to the first step of reconstruction.

```
$\mathsf{Setup}(1^\lambda)$:
    – Pick a prime-order group $\mathcal{G}$ with generator $g$ in which the decisional Diffie-
      Hellman problem is assumed to be hard. Let $p$ be the order of that group.
    – Publish $\mathsf{params} = (\mathcal{G}, p, g)$.
$\mathsf{KeyGen}(\mathsf{params})$:
    – Pick a random $sk \in [p]$.
    – Publish $pk = g^{sk}$.
$\mathsf{Enc}(\mathsf{params}, pk, m \in \mathcal{G})$:
    – Pick a random $y \in [p]$.
    – $u = g^y$.
    – $v = (pk)^y m$.
    – Return $c = (u, v)$.
$\mathsf{Dec}(\mathsf{params}, sk, c = (u, v))$:
    – Return $m = \frac{v}{u^{sk}}$.
$\mathsf{Eval}(\mathsf{params}, c_1 = (u_1, v_1), c_2 = (u_2, c_2), \times)$:
    – $u = u_1 u_2$.
    – $v = v_1 v_2$.
    – Return $c = (u, v)$.
```

Fig. 10: ElGamal Multiplicatively Homomorphic Public Key Encryption Scheme (EG) [ElG84]

```
$\mathsf{Setup}(1^\lambda)$: same as $\mathsf{EG.Setup}$.
$\mathsf{Share}(n, t \leq n, m \in \mathbb{Z}_p)$:
    – Pick a random degree-$t$ polynomial $f$ in $\mathbb{Z}_p$ which has $m$ as its $y$-intercept.
      This can be done by picking $t$ random coefficients $\mathsf{coef}_1, \ldots, \mathsf{coef}_{t-1} \in \mathcal{G}$,
      and setting $f(x) = m + \sum_{j=1}^{t-1} \mathsf{coef}_j x^j$.
    – Return $\{\mathsf{Share}_i\}_{i \in [1,\ldots n]}$ where $\mathsf{Share}_i = (i, g^{f(i)})$.
$\mathsf{Reconstruct}(\{\mathsf{Share}_i = (i, y_i)\}_{i \in \mathcal{R}' \subseteq [n]})$:
    – Perform polynomial interpolation over the shares in the exponent to re-
      cover $g^{\tilde{m}}$. As long as $|\mathcal{R}'| > t$, this can be done by throwing out values in
      $\mathcal{R}'$ until $|\mathcal{R}'| = t + 1$, and doing the following:

          $$g^{\tilde{m}} = \prod_{i \in \mathcal{R}'} y_i^{\prod_{j \in \mathcal{R}', j \neq i} \frac{j}{j-i}}$$

    – Recover $\tilde{m}$ by brute force search.
$\mathsf{Eval}(\mathsf{Share}_i = (i, y_i), \mathsf{Share}'_i = (i, y'_i), +)$: $\mathsf{Share}^+_i = (i, y_i y'_i)$
```

Fig. 11: Exponential Shamir Secret Sharing Scheme (EShamir)

D.4.2 CRT-and-Paillier

We also build share-and-encrypt HATE out of Camenisch-Shoup encryption and Chinese Remainder Theorem based secret sharing. Unlike Shamir-and-ElGamal (Section D.4.1), this HATE allows us to use large message spaces.

CS Additively Homomorphic Encryption. We use a slightly modified version of the Paillier-style verifiable encryption scheme described by Camenisch and Shoup [CS03].[5] Figure 12 describes the this scheme. Our modifications consist solely of removing elements from the ciphertext, so the modified scheme naturally inherits the CPA security of the original (but not its CCA security).[6].

```
KeyGen($1^\lambda$):
    – Let $N = pq$ where $p = 2p' + 1$ and $q = 2q' + 1$, and $p'$ and $q'$ are $\lambda$-bit primes.
    – Let $h = 1 + N$.
    – Choose a random $g' \in Z^*_{N^2}$, and set $g = (g')^{2N} \bmod N^2$. ($g$ is a generator of a size-$p'q'$ subgroup with high probability.)
    – Choose a random secret key $sk \in \{1, \dots, \lfloor (N^2)/4 \rfloor\}$.
    – Return $pk = g^{sk} \bmod N^2$.
Enc($pk, m$):
    – Choose a random $r \in [N/4]$.
    – Return $c = (g^r \bmod N^2, pk^r h^m \bmod N^2)$.
Dec($sk, c = (u, v)$):
    – $z = \frac{v}{u^{sk}} \bmod N^2$. (Note that $z = h^m$ if $c$ is an encryption of $m$.)
    – Return $m' = \frac{z-1}{N}$ with division over integers. (Note that $m'$ is the discrete log of $z$ w.r.t. $h$.)
```

Fig. 12: Camenisch-Shoup Additively Homomorphic Encryption Scheme (CS). We omit Setup, since this scheme does not require setup.

CRT Secret Sharing. We use a classic secret sharing scheme based on the Chinese Remainder Theorem, which allows each party to operate homorphically on shares in a different group. This version is due to Asmuth and Bloom [AB06]. We describe it in Figure 13.

The scheme is perfectly correct. Furthermore, it supports a limited number (currently set to n) of homomorphic additions. The setting of parameters in the setup phase in Figure 13 ensures that $n \cdot A \leq N_+$, where each individual sharing corresponds to a vector of modular reductions of an integer less than A. This means that n sharings, added coordinate-wise, will lead to the reconstruction of an integer less than $n \cdot A$. Every set of more than t shares contains enough information for that reconstruction.

The scheme's statistical security relies on the requirement that each unauthorized set of shares can reconstruct the secret integer $\hat{a}$ only modulo some integer that is (a) relatively prime to N_0 and (b) at most N_-, which itself is at most $\frac{A}{N_0 2^k}$. By the

[5] Their scheme is designed to be secure against chosen ciphertext attacks, which is unnecessary for our purposes.

[6] A similarly modified version of this scheme was used by Cunningham *et al.* [CFY17]

following lemma, those conditions ensure that the view of any unauthorized set is within statistical difference 2^{-k} of uniform.

Lemma 4. *Let a, n, t be positive integers, and let A be uniformly random in $\{a' \in [a] : a' \bmod n = t\}$. Then for all positive integers $m < a$ that are relatively prime to n, the distribution of the random variable $B = A \bmod m$ is within statistical difference $\frac{nm}{a}$ of uniform.*

Proof. Consider the number of $a' \in [a]$ that solve both the equations $a' \bmod n = t$ and $a' \bmod m = u$ (for some u). Since m and n are relatively prime, this system is equivalent to $a' \bmod mn = v$ for some particular v. The number of solutions to this is $\lfloor \frac{a}{mn} \rfloor$ or $\lceil \frac{a}{mn} \rceil$. Thus, the probability that $B = u$ is always with $1 \pm \frac{mn}{a}$ of a uniform element of $\mathbb{Z}_m$. The total variation distance from uniform is thus at most $\frac{mn}{a}$.

Lemma 5. *The CRT secret sharing scheme (*CRTss*) described in Figure 13 is share simulatable.*

Proof. Recall the share simulatability game from Figure 9. On input a set of unauthorized shares $\{\mathsf{Share}_i\}_{i \in \mathcal{C}}$ which were created as a sharing of m_L, and a target message m_R, first find a nonnegative integer $a < N_0 \prod_{i \in \mathcal{C}} N_i$ such that $a \bmod N_0 = m_R$ and $a \bmod N_i = \mathsf{Share}_i$ for $i \in \mathcal{C}$. Such an integer exists since the moduli are all relatively prime. Next, select a random $\hat{a}_R \in [A]$ such that $\hat{a}_R \bmod (N_0 \prod_{i \in \mathcal{C}} N_i) = a$. The correctness condition of the secret sharing scheme implies that $A > N_0 \prod_{i \in \mathcal{C}} N_i$, so this step is always possible. Finally, we produce the new shares as $\mathsf{Share}'_i = \hat{a} \bmod N_i$ for $i \in \mathcal{R} \setminus \mathcal{C}$.

By Lemma 4 above, the distribution of t or fewer shares of m_L are statistically indistinguishable from the corresponding distibution for m_R. The share simulation algorithm above selects a uniformly random sharing of m_R that is consistent with the unauthorized shares of m_L. The joint distribution is therefore statistically close to that of a fresh sharing of m_L.

```
$\mathsf{Share}(n, N_1, ..., N_n, N_0, t \leq n, m \in Z_{N_0}, 1^\lambda)$:
    – Let $\begin{cases} N_+ \overset{\text{def}}{=} \min_{J \subseteq [n] : |J| = t+1} \prod_{i \in J} N_i \\ A \overset{\text{def}}{=} \lfloor N_+/n \rfloor \\ N_- \overset{\text{def}}{=} \max_{J \subseteq [n] : |J| = t} \prod_{i \in J} N_i \end{cases}$.
    – If $N_0 \cdot 2^\lambda \cdot N_- > A$ then stop and return "Error: message space too large
      for $\lambda$ bits of security."
    – Select $a \in_R \{a' \in [A] : a' \bmod N_0 = m\}$
    – Return $(\mathsf{Share}_1, \ldots, \mathsf{Share}_n)$ where $\mathsf{Share}_i = (i, a \bmod N_i)$.
$\mathsf{Reconstruct}(\mathsf{Share}_{i_1} = (i_1, y_{i_1}), \ldots, \mathsf{Share}_{i_t} = (i_t, y_{i_t}))$:
    – Find the unique $\hat{a} \in Z_{\prod_j N_j}$ such that $\hat{a} \equiv y_i \mod N_i$ for all $i \in \{i_1, ..., i_t\}$.
    – Return $\hat{m} = \hat{a} \bmod N_0$.
$\mathsf{Eval}(+, \mathsf{Share}_i = (i, y), \mathsf{Share}'_i = (i, y'))$:          $\mathsf{Share}^+_i = (i, y + y' \bmod N_i)$
```

Fig. 13: Chinese Remainder Secret Sharing Scheme (CRTss). We omit Setup, since this scheme does not require setup.

E Homomorphic Recipient-Compact Obfuscation-Based HATE

In this section, we describe our homomorphic obfuscation-based HATE scheme. The program each sender must obfuscate and include in their public key is described in Algorithm 4. The obfuscation-based HATE is described in Construction 4.

Algorithm 4 $f_{k_w,k_{\mathsf{Share}},k_{\mathsf{Enc}},\mathsf{SIG}.pk}(\overrightarrow{pk} = \{\mathsf{PKE}.pk_j\}_{j\in\mathcal{R}}, \mathsf{idx}, \mathsf{nonce}, \sigma)$

```
**The following values are hardcoded:**
    $\mathsf{params} = (\lambda, n, t)$, where
        $\lambda$ is the security parameter,
        $n$ is the number of recipients, and
        $t$ is the threshold.
    $k_w$, a secret PPRF key used to recover the mask $w$ from nonce nonce
    $k_{\mathsf{Share}}$, a secret PPRF key used to secret share the mask $w$
    $k_{\mathsf{Enc}} = (k_{\mathsf{Enc},1}, \ldots, k_{\mathsf{Enc},n})$, secret PPRF keys used to encrypt shares
    $\mathsf{SIG}.pk$, a signature verification key
**The following values are expected as input:**
    $\overrightarrow{pv} = \{pv_i \in \{0,1\}^{2\lambda}\}_{i\in\mathcal{R}}$, lexicographically ordered public values
    idx, an index
    nonce
    $\sigma$, a signature
**if** $\mathsf{SIG.Verify}(\mathsf{SIG}.pk_{\mathsf{Sndr}}, (\overrightarrow{pk}, \mathsf{nonce}), \sigma)$ **then**
    $w \leftarrow \mathsf{PPRF}_{k_w}(\mathsf{nonce})$
    $r \leftarrow \mathsf{PPRF}_{k_{\mathsf{Share}}}(\mathsf{nonce})$
    $[w]_{\mathsf{idx}} \leftarrow \mathsf{SS.Share}(w, n, t; r)[\mathsf{idx}]$ {This gives the idxth secret share of $w$}
    $r_{\mathsf{Enc},\mathsf{idx}} \leftarrow \mathsf{PPRF}_{k_{\mathsf{Enc},\mathsf{idx}}}(\mathsf{nonce})$
    $c \leftarrow \mathsf{PKE.Enc}(\overrightarrow{pk}_{\mathsf{idx}}, [w]_{\mathsf{idx}}; r_{\mathsf{Enc},\mathsf{idx}})$
    {This gives an encryption of the idxth secret share of $w$. Encryption uses randomness $r_{\mathsf{Enc},\mathsf{idx}}$.}
    **return** $c$
```

Theorem 12 (Restated from Theorem 7). *The modified obfuscation-based ATE (Construction 4) is (n, t)-super-statically secure (Definition 4) for any polynomial n, t, as long as* iO *is a secure indistinguishability obfuscator,* PPRF *is a secure puncturable* PRF*,* SS *is a secure secret sharing scheme,* SIG *is a constrained signature scheme, and* PKE *is a secure public-key encryption scheme. Moreover, it is $\mathcal{F}$-homomorphic if* SS *is $\mathcal{F}$ homomorphic (where $\mathcal{F}$ includes subtraction from a constant), and if* PKE *is $\mathcal{F}'$-homomorphic (where $\mathcal{F}'$ includes the evaluation of $\mathcal{F}$ on* SS *secret shares).*

Proof. In order to prove Theorem 7, we must show the modified obfuscation-based ATE construction is super-statically semantically secure and super-statically partial decryption simulatable. We prove super-static semantic security below, by showing a sequence of indistinguishable games starting at $\mathsf{EXP}(\mathcal{A}, \lambda, n, t, R)$ and ending at a message-independent game. The proof of partial decryption simulatability is the same as for Theorem 1. The homomorphism follows directly from the homomorphisms of the

```
Let the public parameters $\mathsf{params} = (1^\lambda, n, t)$ consist of the security parameter $\lambda$,
the number of recipients $n$, and the threshold $t$.

$\mathsf{KeyGen}(\mathsf{params})$:
    $(pk, sk) \leftarrow \mathsf{PKE.KeyGen}(\mathsf{params})$
    return $(pk, sk)$
$\mathsf{KeyGen_{Sndr}}(\mathsf{params})$: This is exactly as in Construction 1, except that
  the sender generates $n$ additional PPRF keys $k_{\mathsf{Enc},1}, \ldots, k_{\mathsf{Enc},n}$, and instead of
  obfuscating $f$ from Algorithm 1 to get $\mathsf{ObfFunc}$, the sender obfuscates $f$ from
  Algorithm 4
$\mathsf{Enc}(\mathsf{params}, sk_{\mathsf{Sndr}} = (\mathsf{SIG}.sk, k_w), \overrightarrow{pk} = \{\mathsf{PKE}.pk_j\}_{j \in \mathcal{R}, |\mathcal{R}| \geq t}, m)$:
  This is exactly as in Construction 1
$\mathsf{PartDec}(\mathsf{params}, pk_{\mathsf{Sndr}} = \mathsf{ObfFunc}, \{\mathsf{PKE}.pk_j\}_{j \in \mathcal{R}}, \mathsf{PKE}.sk_i, c = (\mathsf{nonce}, e, \sigma))$:

    if the ciphertext $c$ is an output of a homomorphic evaluation then
      $c_m \leftarrow c$
    else
      $c_e \leftarrow \mathsf{PKE.Enc}(\mathsf{params_{PKE}}, \mathsf{PKE}.pk_i, e)$
      Let $\mathsf{idx}$ be the index of the public key corresponding to the secret key
      $\mathsf{PKE}.sk$ in a lexicographic ordering of $\overrightarrow{pk} = \{\mathsf{PKE}.pk_j\}_{j \in \mathcal{R}}$
      $c_w \leftarrow \mathsf{ObfFunc}(\{\mathsf{PKE}.pk_j\}_{j \in \mathcal{R}}, \mathsf{idx}, \mathsf{nonce}, \sigma)$
      Let $f'_-$ be the function which homomorphically evaluates subtraction over
      $\mathcal{G}$ on two $\mathsf{SS}$ secret shares.
      $c_m \leftarrow \mathsf{PKE.Eval}(\mathsf{params_{PKE}}, \mathsf{PKE}.pk_i, (c_e, c_w), f'_-)$
    $[m]_{\mathsf{idx}} \leftarrow \mathsf{PKE.Dec}(\mathsf{params_{PKE}}, \mathsf{PKE}.sk_i, c')$
    $d_i = [m]_{\mathsf{idx}}$
    return $d_i$
$\mathsf{FinalDec}(\{d_i\}_{i \in \mathcal{R}' \subset \mathcal{R}})$:
  Perform exponential Shamir reconstruction $\mathsf{EShamir.Reconstruct}(\{d_i\}_{i \in \mathcal{R}'})$ as
  described in Figure 11 to recover $m$
$\mathsf{Eval}(\{pk_{\mathsf{Sndr}}\}_{\mathsf{Sndr} \in \mathcal{S}}, \overrightarrow{pk} = \{pk_i\}_{i \in \mathcal{R}}, [c_1, \ldots, c_\ell], f)$:
    {Note that this algorithm receives the public keys for all senders Sndr (and
    thus their obfuscated programs). Without loss of generality, let ciphertext
    $c_q = (\mathsf{nonce}_q, e_q, \sigma_q)$ be from sender $P_q$ (and therefore requiring the use of
    $\mathsf{ObfFunc}_q$).}
    for ciphertext indices $q \in [1, \ldots, \ell]$ do
      for receivers $i \in \mathcal{R}$ do
        Let $\mathsf{idx}$ be the index of $\mathsf{PKE}.pk_i$ in a lexicographic ordering of $\overrightarrow{pk}$
        $c_{i,q} \leftarrow \mathsf{ObfFunc}_q(\overrightarrow{pk}, \mathsf{idx}, \mathsf{nonce}_q, \sigma_q))$
      Let $f'_f$ be the function which homomorphically evaluates $f$ on $\ell$ $\mathsf{SS}$ secret
      shares.
      $c^*_i = \mathsf{PKE.Eval}(\mathsf{params_{PKE}}, \mathsf{PKE}.pk_i, [c_{i,1}, \ldots, c_{i,\ell}], f'_f)$
    return $c^* = \{c^*_i\}_{i \in \mathcal{R}}$
```

Construction 4: Obfuscation-Based HATE

underlying secret sharing scheme SS and the underlying public key encryption scheme PKE.

Game 1 This is the same as Game 1 in the proof of Theorem 1. That is, this is $\mathsf{EXP}(\mathcal{A}, \lambda, n, t, R)$ as described in Definition 2 and Figure 1.

Game 2 This is the same as Game 2 in the proof of Theorem 1. That is, in this game, when creating the sender's public key (specifically, the sender's signature verification key $\mathsf{SIG}.pk$), the challenger Chal generates a constrained verification key instead of a regular one. The constraint is designed to ensure that the challenge nonce nonce^* on which the PPRFs are called can only ever be associated with the challenge recipient set.

Game 2 is indistinguishable from Game 1 by the security of constrained signatures.

Game 3 This is the same as Game 4 in the proof of Theorem 1, except that in addition to puncturing the PPRF keys k_w and k_{Share}, the challenger Chal also punctures $k_{\mathsf{Enc},\mathsf{idx}}$ for all $\mathsf{idx} \in [n]$. To preserve the input-output behavior of the program, Chal computes $r^*_{\mathsf{Enc},\mathsf{idx}} = \mathsf{PPRF}_{k_{\mathsf{Enc},\mathsf{idx}}}(\mathsf{nonce}^*)$, and modifies the program to set $r_{\mathsf{Enc},\mathsf{idx}} = r^*_{\mathsf{Enc},\mathsf{idx}}$ when $\mathsf{nonce} = \mathsf{nonce}^*$, as shown in Algorithm 5.

Algorithm 5 $f^{\mathsf{Game\ 3}}_{k_w\{\mathsf{nonce}^*\},k_{\mathsf{Share}}\{\mathsf{nonce}^*\},k_{\mathsf{Enc}}\{\mathsf{nonce}^*\},\mathsf{SIG}.pk,\mathsf{nonce}^*,w^*,r^*,r^*_{\mathsf{Enc},1},\ldots,r^*_{\mathsf{Enc},n}}(\overrightarrow{pk}, \mathsf{idx}, \mathsf{nonce}, \sigma)$

```
if $\mathsf{SIG.Verify}(\mathsf{SIG}.pk_{\mathsf{Sndr}}, (\overrightarrow{pk}, \mathsf{nonce}), \sigma)$ then
    if $\mathsf{nonce} = \mathsf{nonce}^*$ then
        $w \leftarrow w^*$
        $r \leftarrow r^*$
        $r_{\mathsf{Enc},\mathsf{idx}} \leftarrow r^*_{\mathsf{Enc},\mathsf{idx}}$
    else
        $w \leftarrow \mathsf{PPRF}_{k_w}(\mathsf{nonce})$
        $r \leftarrow \mathsf{PPRF}_{k_{\mathsf{Share}}}(\mathsf{nonce})$
        $r_{\mathsf{Enc},\mathsf{idx}} \leftarrow \mathsf{PPRF}_{k_{\mathsf{Enc},\mathsf{idx}}}(\mathsf{nonce})$
    $[w]_{\mathsf{idx}} \leftarrow \mathsf{SS.Share}(w, n, t; r)[\mathsf{idx}]$ {This gives the idxth secret share of $w$}
    $c \leftarrow \mathsf{PKE.Enc}(\overrightarrow{pk}_{\mathsf{idx}}, [w]_{\mathsf{idx}}; r_{\mathsf{Enc},\mathsf{idx}})$
    {This gives an encryption of the idxth secret share of $w$. Encryption uses randomness $r_{\mathsf{Enc},\mathsf{idx}}$.}
    return $c$
```

Game 3 is indistinguishable from Game 2 by the security of indistinguishability obfuscation; the programs have identical input-output behavior.

Game 4 This game is the same as Game 5 in the proof of Theorem 1. That is, in this game, the challenger Chal chooses r^* truly at random.

Game 4 is indistinguishable from Game 3 by the security of puncturable PRFs.

Game 5 This game is the same as Game 6 in the proof of Theorem 1. That is, in this game, the challenger Chal chooses w^* truly at random.

Game 5 is indistinguishable from Game 4 by the security of puncturable PRFs.

Game 6.idx **for** $\mathsf{idx} \in [1, \ldots, n]$

In this game, the challenger Chal chooses the encryption randomness $r_{\mathsf{Enc},\mathsf{idx}}$ uniformly at random (instead of setting it to $k_{\mathsf{Enc},\mathsf{idx}}(\mathsf{nonce}^*)$).

Game 6.1 is indistinguishable from Game 5, and Game 6.idx is indistinguishable from Game 6.($\mathsf{idx} - 1$) for $\mathsf{idx} \in [2, \ldots, n-t]$, by the security of puncturable PRFs. We let Game 6 denote Game 6.n.

Game 7 In this game, the challenger Chal modifies the obfuscated program to hardcode the ciphertexts $(c_1^*, \ldots, c_n^*)$ instead of hardcoding w^*, r^* and $r^*_{\mathsf{Enc},1}, \ldots, r^*_{\mathsf{Enc},n}$, as described in Algorithm 6. To preserve the input-output behavior of the program, Chal computes c^*_{idx} exactly as it would have been computed in Algorithm 5.

Algorithm 6 $f^{\mathsf{Game\ 7}}_{k_w\{\mathsf{nonce}^*\},k_{\mathsf{Share}}\{\mathsf{nonce}^*\},k_{\mathsf{Enc}}\{\mathsf{nonce}^*\},\mathsf{SIG}.pk,\mathsf{nonce}^*,c_1^*,\ldots,c_n^*}(\overrightarrow{pk}, \mathsf{idx}, \mathsf{nonce}, \sigma)$

```
if $\mathsf{SIG.Verify}(\mathsf{SIG}.pk_{\mathsf{Sndr}}, (\overrightarrow{pk}, \mathsf{nonce}), \sigma)$ then
  if $\mathsf{nonce} = \mathsf{nonce}^*$ then
    $c \leftarrow c^*_{\mathsf{idx}}$
  else
    $w \leftarrow \mathsf{PPRF}_{k_w}(\mathsf{nonce})$
    $r \leftarrow \mathsf{PPRF}_{k_{\mathsf{Share}}}(\mathsf{nonce})$
    $r_{\mathsf{Enc,idx}} \leftarrow \mathsf{PPRF}_{k_{\mathsf{Enc,idx}}}(\mathsf{nonce})$
    $[w]_{\mathsf{idx}} \leftarrow \mathsf{SS.Share}(w, n, t; r)[\mathsf{idx}]$ {This gives the idxth secret share of $w$}
    $c \leftarrow \mathsf{PKE.Enc}(\overrightarrow{pk}_{\mathsf{idx}}, [w]_{\mathsf{idx}}; r_{\mathsf{Enc,idx}})$
    {This gives an encryption of the idxth secret share of $w$. Encryption uses randomness $r_{\mathsf{Enc,idx}}$.}
  return $c$
```

Game 7 is indistinguishable from Game 6 by the security of indistinguishability obfuscation; the programs have identical input-output behavior.

Game 8.i for $i \in [1, \ldots, n-t]$
In this game, the challenger Chal modifies the obfuscated program to hardcode encryptions of zero for honest parties. Let idx_i be the index of the ith honest public key in a lexicographic ordering of $\overrightarrow{pk}^*$. Chal sets $c^*_{\mathsf{idx}_i} \leftarrow \mathsf{PKE.Enc}(\overrightarrow{pk}^*_{\mathsf{idx}_i}, 0; r^*_{\mathsf{idx}})$.
Game 8.1 is indistinguishable from Game 7, and Game 8.i is indistinguishable from Game 8.($i-1$) for $i \in [2, \ldots, n-t]$, by the semantic security of PKE.
We let Game 8 denote Game 8.($n-t$).

Game 9 This game is the same as Game 8 in the proof of Theorem 1. That is, in this game, the challenger Chal encrypts shares of 0 for corrupt parties. Let idx_i be the index of the ith corrupt public key in a lexicographic ordering of $\overrightarrow{pk}^*$. Instead of $[w^*]_{\mathsf{idx}_i} \leftarrow \mathsf{SS.Share}(w^*, n, t; r^*)[\mathsf{idx}_i]$, Chal sets $[w^*]_{\mathsf{idx}_i} \leftarrow \mathsf{SS.Share}(0, n, t; r^*)[\mathsf{idx}_i]$.
Game 9 is indistinguishable from Game 8 by the privacy property of SS.

Game 10 In this game, the challenger switches to using a random message m^*. Game 10 is indistinguishable from Game 9 because the distributions do not change at all; e^* is still uniformly random, and the obfuscated program, which no longer contains any information about w^*, is unaffected.

The rest of the games are what we did before, but in reverse, with m_L instead of m_R.

Game	Justification	$\mathsf{SIG}.pk$	Obfuscated Program	c^*	m^*
1		real	real	real	m_R
2	Constrained Signatures	constrained to only verify on nonce^* when $\{pv_j\}_{j\in\mathcal{R}} = \{pv_j\}_{j\in\mathcal{R}^*}$			
3	iO		puncture k_w, k_{Share} and k_{Enc} at nonce^*; hardcode correct values w^*, r^* and $\{r^*_{\mathsf{Enc},\mathsf{idx}}\}_{\mathsf{idx}\in[1,\ldots,n]}$		
4	PPRF		hardcode random r^*		
5	PPRF		hardcode random mask w^*	compute e^* using the random mask	
6	PPRF		hardcode random $\{r^*_{\mathsf{Enc},\mathsf{idx}}\}_{\mathsf{idx}\in[1,\ldots,n]}$		
7	iO		hardcode $\{c^*_{\mathsf{idx}}\}_{\mathsf{idx}\in[1,\ldots,n]}$ instead of w^*, r^* and $\{r^*_{\mathsf{Enc},\mathsf{idx}}\}_{\mathsf{idx}\in[1,\ldots,n]}$		
8	PKE		hardcode encryptions of 0 instead of encryptions of shares of w^* for honest parties		
9	SS		hardcode encryptions of shares of 0 instead of encryptions of shares of w^* for corrupt parties		
10	identical distributions				0

Fig. 14: Summary of Hybrids in Proof of Theorem 7