Threshold Fully Homomorphic Encryption and Secure Computation

Steven Myers Mona Sergi abhi shelat*

Abstract

Cramer, Damgård, and Nielsen [CDN01] show how to construct an efficient secure multi-party computation scheme using a threshold homomorphic encryption scheme that has four properties i) a honest-verifier zero-knowledge proof of knowledge of encrypted values, ii) proving multiplications correct iii) threshold decryption and iv) trusted shared key setup. Naor and Nissim [NN01a] show how to construct secure multi-party protocols for a function f whose communication is proportional to the communication required to evaluate f without security, albeit at the cost of computation that might be exponential in the description of f.

Gentry [Gen09a] shows how to combine both ideas with fully homomorphic encryption in order to construct secure multi-party protocol that allows evaluation of a function f using communication that is **independent of the circuit description of** f and computation that is polynomial in $|f|$. This paper addresses the major drawback's of Gentry's approach: we eliminate the use of non-black box methods that are inherent in Naor and Nissim's compiler.

To do this we show how to modify the fully homomorphic encryption construction of van Dijk et al. [vDGHV10] to be threshold fully homomorphic encryption schemes. We directly construct (information theoretically) secure protocols for sharing the secret key for our threshold scheme (thereby removing the setup assumptions) and for jointly decrypting a bit. All of these constructions are constant round and we thoroughly analyze their complexity; they address requirements (iii) and (iv). The fact that the encryption scheme is fully homomorphic addresses requirement (ii).

To address the need for an honest-verifier zero-knowledge proof of knowledge of encrypted values, we instead argue that a weaker solution suffices. We provide a 2-round blackbox protocol that allows us to prove knowledge of encrypted bits. Our protocol is not zero-knowledge, but it provably does not release any information about the bit being discussed, and this is sufficient to prove the correctness of a simulation in a method similar to Cramer et al.

Altogether, *we construct the first black-box secure multi-party computation protocol that allows evaluation of a function f using communication that is independent of the circuit description of f.*

Keywords: Fully Homomorphic Encryption, Threshold Encryption, Secure Multi-Party Communication, Communication and Round Complexity, Proof Of Knowledge

*The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211.

1 Introduction

The first goal of this paper is to construct a *threshold* fully homomorphic public key encryption scheme (FHE) in which the decryption key is shared among n players. Any k of the n players can collaborate to decrypt a ciphertext that is encrypted under the public key of the scheme. We describe schemes for distributed key generation, encryption and methods to prove knowledge of the encrypted value, and finally distributed decryption for a fully homomorphic encryption scheme based on the scheme presented by van Dijk et al [vDGHV10]. Our protocols run in a small constant number of rounds and we carefully analyze their communication complexity. For example, our decryption procedure for the Approximate-GCD based scheme requires 4 to 9 rounds of communication (depending on the parameter settings).

Following ideas from Cramer, Damgård, and Nielson [CDN01] and Gentry [Gen09a], our threshold FHE scheme allows for construction of a secure multiparty computation protocol for n players that tolerates up to $n/2$ malicious parties and has nearly optimal communication parameters. In particular, the communication complexity of our protocol is independent of the circuit size of the function being computed. Although Gentry points out that this result follows directly from the existence of FHE, the existence of any secure multiparty computation, and the compiler of Naor and Nissim [NN01b] (which relies on the existence of PCPs), our use of threshold FHE avoids the non-blackbox construction methods inherent in this combination. For example, our shared FHE decryption procedure is more efficient than using general secure function evaluation techniques to decrypt an FHE ciphertext.

Related work The notion of threshold cryptography scheme was implicitly motivated by Shamir in [Sha79], and was formally introduced by Desmedt et al. [Des87]. Several extensions and schemes have been considered in the literature for different public encryption and signature schemes, including but not limited to [DF89] for threshold ElGammal scheme [ElG85], [Rab98, FGMY97a, FGMY97b] for threshold RSA, and [DJ01] for a variant of Paillier's scheme. Cramer, Damgård and Nielson [CDN01], along with Jakobbsson and Juels [JJ00] show how to use threshold cryptography to construct secure multiparty computation protocols. In more detail, [CDN01] showed homomorphic threshold cryptosystem can be used to achieve general multiparty computation protocols which is secure against active adversaries.

1.1 Our Techniques

We present a threshold scheme for fully homomorphic encryption based on the construction of [vDGHV10]. We find that many known techniques for secret sharing can be used to efficiently implement the key generation and decryption methods. In a few cases, we modify the construction from [vDGHV10] in simple ways that allow for much faster secure distributed implementation. For example, we relax the constraints on the Hamming weight for some portions of the secret key of the FHE scheme. Following the construction of FHE based on Gentry's paradigm, our distributed threshold key generation protocol needs to produce encryptions of each bit of the secret key. Thus, we must also implement a method for shared encryption of bits. Finally, in order to completely avoid generic non-blackbox zero-knowledge and generic SMC techniques, we present a black-box proof of knowledge of plaintext for any FHE scheme with circuit privacy. This construction relies on the combination of recent results concerning selective-opening security along with a novel application of a verifiable secret sharing scheme. In the final section of the paper, we illustrate how to combine these primitives into a secure multiparty protocol that tolerates a dishonest minority and has communication complexity that is polynomial in n—the number of players—and the security parameter λ—i.e., it does not depend on the circuit C being evaluated.

2 Preliminaries and Notation

Basic Notations We define some notation. If S is a set, let $s \leftarrow S$ denote the process of selecting the element s uniformly at random from S. Given two families of distributions $\{\mathcal{D}_{1i}\}_i \in \mathbb{N}$ and $\{\mathcal{D}_{2i}\}_i \in \mathbb{N}$, we say that they are statistically close, denoted $\mathcal{D}_1 \approx_s \mathcal{D}_2$, if $\Delta(\mathcal{D}_{1i}, \mathcal{D}_{2i}) \leq 1/poly(i)$ for all polynomials and sufficiently large i, where $\Delta(a, b)$ is the statistical distance between distributions a and b. We denote the shares of a held by adversary as $[a]^I$.

A 3-tuple of algorithms (**Gen**, **Enc**, **Dec**) is a threshold fully homomorphic encryption scheme if the following holds.

Key Generation There exists an n-party protocol **Gen** that at each invocation returns a new public key PK and the secret key $(\mathrm{SK}_1, \ldots, \mathrm{SK}_n)$, where SK_i is the share of the secret key for Player_i.

Encryption There exists a PPT algorithm $\mathbf{Enc}_{\mathrm{PK}}(m, r)$, returns the encryption of the plaintext m under the public key PK with random coins r.

Decryption There exists a n-party secure protocol $\mathbf{Dec}(c, \mathrm{SK}_1, \ldots, \mathrm{SK}_n)$, which returns the plaintext m using the shares SK_i held by honest party Player_i, where $c = \mathbf{Enc}(m, r)$ for some random r.

$f_{\mathbf{PK}}$-homomorphic There exist a PPT algorithm which given a polynomial f, ciphertexts $c_1 \in \mathbf{Enc}_{\mathrm{PK}}(m_1), \ldots, c_k \in \mathbf{Enc}_{\mathrm{PK}}(m_k)$ for some k and a public key PK, outputs $c \in \mathbf{Enc}(f(m_1, \ldots, m_k))$.

The natural notion of indistinguishability security needs to be modified in the venue of threshold cryptography to take in to account the fact that the adversary has access to shares of the secret-key, and we need to ensure these do not aid it. The appropriate corresponding and natural definition is given below:

Definition 2.1 (Threshold Indistinguishability [CDN01]). *Let A be an efficient adversary, which on input 1^k, a public-key PK, and any subset C of the corresponding secret-key shares $SK_1, \ldots, SK_n$ where $|C| < n/2$, outputs two messages m_0, m_1 and state information σ. Let $(s, c_i) \leftarrow X_i(k, C)$ denote the distribution over (s, c_i) where $(pk, sk_1, \ldots, sk_n)$ are generated by the execution of the key generation protocol, $(m_0, m_1, s) \leftarrow A(1^k, pk, C, \{sk_c\}_{c \in C})$, and $c_i \leftarrow$ **Enc**(pk, m_i). Then $X_i = \{X_i(k, C)\}_{k \in \mathbb{N}, C}$ for $i \in \{0, 1\}$ are ensembles, and we require that $X_0 \approx_c X_1$.*

Next we present a definition of circular threshold security for a semantically secure encryption scheme. Our definition is based on the circular security definition in [GH110], but we must take in to account the fact that the adversary has access to shares of the secret-key, $[\mathrm{SK}]^I$, where $[\mathrm{SK}]^I$ denotes the set of shares of the secret key the adversary has access to. The definition needs to ensure that these shares do not affect the circular security.

Definition 2.2 (Circular Threshold Security)**.** *Let $\Pi = (\textbf{\textit{Gen}}, \textbf{\textit{Enc}}, \textbf{\textit{Dec}})$ to be a semantically secure threshold encryption scheme. For algorithm A and random $k \in \mathbb{N}$, let $\mathsf{IND\text{-}CirThrCPA}_b(\Pi, A, k)$ to be the output of the following experiment:*

```
$\mathsf{IND\text{-}CirThrCPA}_b(\Pi, A, k)$
    $(PK, SK) \leftarrow \textbf{\textit{Gen}}(1^k)$
    $m_0 \leftarrow SK$
    $m_1 \leftarrow 0^{|SK|}$
    $c^* \leftarrow \textbf{\textit{Enc}}(PK, m_b)$
    $\textit{Output } A([SK]^I, PK, c^*)$
```

Encryption scheme Π is circular threshold secure if for all p.p.t algorithms A it holds that the following two ensembles are computationally indistinguishable:

$$\{\mathsf{IND\text{-}CirThrCPA}_0(\Pi, A, k)\}_k \approx_c \{\mathsf{IND\text{-}CirThrCPA}_1(\Pi, A, k)\}_k$$

3 Proof of Knowledge of an Encryption

As noted in the Introduction, the method of Cramer, Damgård, and Nielsen [CDN01] requires an honest-verifier zero-knowledge proof of knowledge of encrypted values for the threshold schemes that they employ.

We provide a weaker solution to the final requirement. We provide a 2-round proof of knowledge of encrypted bits, which alas, is not zero-knowledge. However, the protocol provably does not release any information about the bit being discussed, and this is sufficient to prove the correctness of a simulation in a method similar to Cramer et al. Moreover, our construction only makes black-box use of the underlying FHE scheme, and works for any circuit-private FHE scheme (actually, it works for a selective-opening secure scheme).

We construct this proof through a two-step process. At a high-level, instead of encrypting a bit b, we will use a specific $\binom{n}{n/2+2}$ verifiable secret sharing scheme to generate n shares of b and encrypt those shares. In order to give a proof of knowledge of the encryption of b, we will allow a verifier to select $n/2 + 1$ of the encryptions of shares of b, and then direct the Prover to decommit them by revealing the randomness used to encrypt them. To extract the bit, our extractor rewinds the proof and selects an alternate $n/2 + 1$ shares, so that with high probability, it can use $n/2 + 1$ shares to reconstruct b, and only b due to the verifiability of the secret sharing scheme. The problem with this approach is that revealing the randomness for an encryption raises selective decommitment issues. We use techniques from Hemenway et al. [HLOV09] to construct a bit-wise Indistinguishable Selective-Opening Secure encryption scheme from our threshold fully-homomorphic scheme.

We note that the construction of our bit-wise Indistinguishable Selective-Opening Secure scheme does not constitute a fully homomorphic encryption scheme. For example, we cannot add two sets of shares encoding b_0 and b_1 and expect the new shares to encode $b_0 + b_1$. However, this is not a problem. We can homomorphically evaluate the reveal function of the secret sharing scheme to get a single encryption representing the reconstituted bit. This encryption can then be used to homomorphically evaluate the function as in Cramer et al. [CDN01].

Using FHE to construct a Selective Opening Encryption Scheme Hemenway et al. [HLOV09] show any re-randomizable encryption scheme can be used to construct a natural lossy encryption scheme and thus, by the result of Bellare et al. [BHY09], is secure against indistinguishable selective opening attacks (IND-SO-ENC). We give the definitions of Lossy encryption and IND-SO-ENC secure encryption below. We note that IND-CPA security is implied by the definition of Lossy encryption, see [HLOV09] for details.

Definition 3.1 (Lossy Encryption)**.** *A lossy public-key encryption scheme is a triple (G, E, D) of P.P.T. algorithms such that:*

Correctness of Injective Keys: *For all $(PK, SK) \leftarrow G(1^k, \mathsf{INJ})$, $b \in \{0,1\}$, and random strings r: $D(SK, E(PK, b, r)) = b$.*

Lossiness of Lossy Keys: *For all $(PK, SK) \leftarrow G(1^k, \mathsf{LOSSY})$: $E(PK, 0) \approx_s E(PK, 1)$.*

Computational Indistinguishability of Lossy and Injective Keys: *Define $s : (x, y) \mapsto x$ to project pairs of public- and secret-keys to pulblic-keys.*

$$\{s(G(1^k, \mathsf{INJ}))\}_k \approx_c \{s(G(1^k, \mathsf{LOSSY}))\}_k$$

Openability: *The following is implied by lossiness. There exists an algorithm (not necessarily poly-time)* Opener *which when given a lossy public-key PK, $(PK, SK \leftarrow G(1^k, \mathsf{LOSSY})$ and a ciphertext $c \leftarrow E(PK, b)$, will with non-negligible probability (over the choice of PK, SK and random bits used to generate c) output two strings r_0, r_1, such that $E(PK, 0, r_0) = c$ and $E(PK, 1, r_1) = c$.*

Thus lossy encryption provides the ability to generate keys that produce "lossy" encryptions in which the distribution of encryptions of 0, is statistically close to the distribution of encryptions of 1. These keys are indistinguishable from regular, injective, keys. An alternate notion of security for encryption relates to the selective opening problem. In this, somewhat counterintuitive, notion we are asked that if there is a sequence of ciphertexts corresponding to different messages being encrypted, can the release of the randomness used to encrypt some of the ciphertexts be used to break the secrecy security of the other ciphertexts. A scheme that is secure against selective openings ensures that the release of randomness used to encrypt some ciphertexts cannot be used to compromise the security of other encryptions with the same public-key.

Definition 3.2 (IND-SO-SEC Encryption Security)**.** *A public-key encryption scheme $\Pi = (G, E, D)$ is Indistinguishable Selective Opening secure if, for any message sampler M that supports efficient conditional resampling, and any P.P.T. adversary $A = (A_1, A_2)$ there exists a negligible function μ such that for all sufficiently large k:*

$$\left|\Pr[A_{\Pi}^{\mathsf{Ind\text{-}SO\text{-}Real}}(1^k) = 1] - \Pr[A_{\Pi}^{\mathsf{Ind\text{-}SO\text{-}Ideal}}(1^k) = 1]\right| \leq \mu(k).$$

A message sampler M is a PPT algorithm that outputs a vector $\vec{m}$ of n messages from a given distribution. It is an efficient conditional resampler if, when given two auxiliary inputs a set of indices $I \subseteq [n]$ and a vector of messages $\vec{m} = (m_1, ..., m_n)$, M will sample another vector $\vec{m'} = (m'_1, \cdots, m'_n)$ conditioned on $m_i = m'_i$ for each $i \in I$. We define the experiments Ind-SO-Real *and* Ind-SO-Ideal *as follows.*

Ind-SO-Real$(1^k, A)$	Ind-SO-Ideal$(1^k, A)$
$(PK, SK) \leftarrow G(1^k)$	$(PK, SK) \leftarrow G(1^k)$
$\vec{m} = (m_1, \ldots, m_n) \leftarrow M$	$\vec{m} = (m_1, \ldots, m_n) \leftarrow M$
$r_1, \ldots, r_n \leftarrow R$	$r_1, \ldots, r_n \leftarrow R$
$(I, \sigma) \leftarrow A_1(PK, E_{PK}(m_1, r_1), \ldots, E_{PK}(m_n, r_n)$	$(I, \sigma) \leftarrow A_1(PK, E_{PK}(m_1, r_1), \ldots, E_{PK}(m_n, r_n)$
$b = A_2(\sigma, (m_i, r_i)_{i \in I}, \vec{m})$	$\vec{m'} = (m'_1, \ldots, m'_n) \leftarrow M_{\vert I, m[I]}$.
	$b = A_2(\sigma, (m_i, r_i)_{i \in I}, \vec{m'})$

Hemenway et al. suggest that the distribution of a "fresh" encryption of a message should be statistically close to a rerandomization of a fixed message. They point out that all homomorphic encryption schemes up to that point achieved this property by adding an encryption of 0 to the current message. While this property was true of all schemes at the time, it is not actually true of the known fully homomorphic encryption schemes, because each time we add an encrypted message to another we increase the amount of noise that is embedded in the ciphertexts, and thus fresh encryptions have less noise than encryptions that have had operations (such as addition) applied to them. Fortunately, the property they state is overly strong, and a simple observation shows that for their construction to go through they only require that the distributions

$$\{r \leftarrow R : E(pk, 0, r) \oplus E(pk, m, r_0)\} \approx_s \{r \leftarrow R : E(pk, 0, r) \oplus E(pk, m, r_1)\},$$

for all public-keys PK, messages m and random strings r_0 and r_1. However, it is simple to see that even these two distributions are not statistically close for the fully homomorphic encryption schemes that have been proposed. Fortunately, both schemes under consideration have rerandomization functions built to ensure *Circuit-Privacy*, similar to what is defined in Def 3.

Definition 3.3. *((Statistical) Circuit Private Homomorphic Encryption). A homomorphic encryption scheme ε is circuit-private for circuits in a set C_ε if, for any key pair (PK, SK) output by **Gen**(λ), any circuit $C \in C_\varepsilon$, and any fixed ciphertext $\psi = \langle \psi_1, \ldots, \psi_t \rangle$ that are in the image of **Enc**$_\varepsilon$ for plaintexts $\pi_1, \ldots, \pi_t$, the following distributions (over the random coins in **Enc**$_\varepsilon$, **Eval**$_\varepsilon$) are (statistically) indistinguishable:*

$$\textbf{\textit{Enc}}_\varepsilon(PK, C(\pi_1, \ldots, \pi_t)) \approx \textbf{\textit{Eval}}_\varepsilon(PK, C, \psi)$$

In the original schemes first presented by both Dijk et al. [vDGHV10] and Gentry [Gen09a], the initial evaluation functions are deterministic and not circuit-private. In order to overcome this problem, both works introduce a method for adding random noise to encryptions, whether they are output from **Eval** or **Enc**, and thus in some sense rerandomizing them. This is done by adding an 'encryption' of 0 to the ciphertext in question, but where the 'encryption' has significantly more noise than would be generated by either the legitimate encryption or evaluation process. Specifically, they introduce P.P.T. algorithms labeled CircuitPrivacy : $\mathcal{C}_b \rightarrow \mathcal{C}'_b$, where C consists of all the ciphertexts that are output from $\textbf{Enc}_{\text{PK}}(b)$ or a call to **Eval** with an encrypted output bit of b. It is the case that for any b and any $c_{b,0}, c_{b,1} \in \mathcal{C}_b$.

$$\text{CircuitPrivacy}(c_{b,0}) \approx_s \text{CircuitPrivacy}(c_{b,1}).$$

In the case of the construction based on approximate GCD, CircuitPrivacy(c) chooses a random subset $S \subseteq \{1, \ldots, \tau\}$, and $r \leftarrow [-2^{\eta-6}, 2^{\eta-6}]$ and output $c' \leftarrow [c + \sum_{i \in S} x_i]_{x_0} + 2r$ (See Appendix C of [vDGHV10] for more details). Gentry describes a similar method for achieving circuit privacy on lattice based encryptions [Gen09a].

We describe our version of Hemenway et al.'s construction below. It takes a homomorphic encryption scheme (G, E, D) with the homomorphism $E(\mathrm{PK}, m_0 + m_1) = E(\mathrm{PK}, m_0) \oplus E(\mathrm{PK}, m_1)$. They define $\mathsf{ReRand} : \mathcal{C} \to \mathcal{C}$ as a rerandomization function evaluated as $c \mapsto c \oplus E(\mathrm{PK}, 0)$.

Key Generation $G'(1^k, b), b \in \{\mathsf{INJ}, \mathsf{LOSSY}\}$**:** Let $(\mathrm{PK}, \mathrm{SK}) \leftarrow G(1^k), c_0 \leftarrow E(\mathrm{PK}, 0), c_1 \leftarrow E(\mathrm{PK}, 1)$ and $c_1' = \leftarrow E(\mathrm{PK}, 0)$. If $b = \mathsf{INJ}$ Output $\mathrm{PK}' = (pk, c_0, c_1)$ and $\mathrm{SK}' = \mathrm{SK}$, else when $b = \mathsf{LOSSY}$ output $\mathrm{PK}' = (\mathrm{PK}, c_0, c_1')$ and $\mathrm{SK}' = \mathrm{SK}$.

Encryption $E'(\mathbf{PK}' = (\mathbf{PK}, c_0, c_1), b)$**:** Output $\mathsf{ReRand}(c_b)$.

Decryption $D'(\mathbf{SK}, c)$**:** Output $D(\mathrm{SK}, c)$.

Theorem 3.4. *If (G, E, D) is a circuit-private FHE, then the blackbox construction (G', E', D') described above is an IND-SO-SEC secure encryption scheme.*

Proof. Follows from [HLOV09] and [BHY09]. □

3.1 Proof of Knowledge

Again, in order to be able to provide a proof of knowledge that the a party has knowledge of the value encrypted, we need to provide a POK. This is done by providing a POK which does not leak information about the bit that is encrypted, but which is not zero-knowledge nor witness indistinguishable. The POK is constructed by encrypting shares of a verifiable secret sharing scheme, where the bit to be encrypted is the secret in question, as opposed to encrypting the bit directly. The encryptions of shares act as commitments. In order to prove knowledge, we decommit some shares by releasing the randomness used to perform the encryption. We release few enough shares to ensure that no information about the underlying bit is released, yet enough that if the prover successfully decommits its challenge, then we are all but sure the prover could decommit enough shares to release the secret-bit.

3.1.1 Verifiable Secret-Sharing Scheme

A $\binom{n}{n/2+2}$ Verifiable Secret-Sharing scheme consists of a sharing algorithm which takes as input a secret s and produces n-shares $s_1, ..., s_n$. These shares have the property that for any $T \subset \{1, \ldots n\}$, $|T| < n/2 + 2$ it is the case that $\{s_i\}_{i \in T}$ is information theoretically independent from s. However, for any $S \subseteq \{1, \ldots n\}$, $|S| \geq n/2 + 2$, it is the case that the reveal algorithm, when given $\{s_i\}_{i \in S}$, can reconstruct s. In a traditional interactive setting we require that all non-cheating parties agree on the reconstructed secret.

We use a modification of the Cramer et al. [CDD+99] verifiable secret sharing scheme. We note that in our application, we do not need to deal with interactive adversaries, nor players, so the scheme is significantly simplified. We present the sharing and revealing algorithms below. It is assumed that all of the operations are in some finite-field F of appropriate size.

Protocol 1 . $[\vec{s}]$, VSShare(s)

```
1: Choose a random degree $n/2 + 1$ bi-variate polynomial $f$ such that $f(0, 0) = s$.
2: Share $s_i = (\vec{a}, \vec{b}) = (i, (f(i, 1), \ldots, f(i, n)), (f(1, i), \ldots, f(n, i)))$.
```

```
3: Output $s_1, ..., s_n$.
   **Protocol 2 .** $[\vec{s}], VSReveal_{\binom{n}{n/2+2}}(s_1, ..., s_{n/2+2})$
1: For each $s_i = (i, \vec{a}_i, \vec{b}_i)$ ensure that $a_i$ and $b_i$ are $n/2+2$-consistent
2: If not output $\perp$.
3: For each $i \neq j$ ensure $s_j, s_i$ are pairwise-consistent
4: If not output $\perp$.
5: Interpolate $f$, based on shares.
6: Output $f(0,0)$
```

Definition 3.5. *A vector $(e_1, ..., e_n) \in F_n$ is $n/2+2-$*consistent *if there exists a polynomial w of degree at most $n/2+1$ such that $w(i) = e_i$ for $0 \leq i < n$.*

Definition 3.6. *Given two shares $s_i = (i, \vec{a}_i = (a_{i1}, \ldots, a_{in}), \vec{b}_i = (b_{1i}, \ldots, b_{ni}))$ and $s_j = (j, \vec{a}_j(a_{j1}, \ldots, a_{jn}), \vec{b}_j = (b_{1j}, \ldots, b_{nj}))$, we say that they are* pairwise consistent *if $a_{ij} = b_{ij}$ and $a_{ji} = b_{ji}$.*

Definition 3.7. *For our purposes it is useful to note that given the $n \times n$ matrix*

$$\begin{bmatrix} f(1,1) & f(1,2) & \ldots & f(1,n) \\ f(2,1) & f(2,2) & \ldots & f(2,n) \\ \vdots & \vdots & \ddots & \vdots \\ f(n,1) & f(n,2) & \ldots & f(n,n) \end{bmatrix},$$

that a share s_i simply corresponds to the i^{th} row and column of the matrix. We will call this the matrix representation of the shares. Notice that when given in the matrix representation, any two shares are necessarily pairwise consistent. Given a set of n pairwise consistent shares $\vec{s} = (s_1, ..., s_n)$, we define $M_{\vec{s}}$ as the $n \times n$ matrix representation of the shares.

Definition 3.8. *We say an $n \times n$ matrix representation of shares has $t-$ consistent indices, if there is a set S, $|S| = t$, such that for each $i \in S$, each row i and column i is $n/2+2$ consistent.*

Lemma 3.9. *Let M be $n \times n$ matrix representation of shares. Let $S, T \subseteq \{1, \ldots, n\}$, $|S| = |T| = n/2+2$, $S \neq T$, and the rows $R_S = \{r_i\}_{i \in S}$, $R_T = \{r_i\}_{i \in T}$ and columns $C_S = \{c_i\}_{i \in S}$, $C_T = \{c_i\}_{i \in T}$ are all $n/2+2-$consistent. Let $s = (s_1, ..., s_{n/2+2})$ and $t = (t_1, ..., t_{n/2+2})$ be the shares drawn from M corresponding to the sets of indices S and T respectively. Then $VSReveal_{\binom{n}{n/2+2}}(s_1, ..., s_{n/2+1}) = VSReveal_{\binom{n}{n/2+2}}(t_1, ..., t_{n/2+1})$.*

Proof. Note that in $VSReveal_{\binom{n}{n/2+2}}$ lines 1–4 will never output $\perp$ under our conditions, so all that we need do is show that f will interpolate to the same value in both cases.

We know that the rows $R_T\{r_i\}_{i \in T}$ and columns $C_R\{c_i\}_{i \in T}$ are all $(n/2+2)-$consistent. Choose any $j \in S \setminus T$. Let $T = \{t_1, \ldots, t_{n/2+2}\}$. Consider $c_j = (c_{1,j}, c_{2,j}, \ldots, c_{n,j})^T$. Since c_j is $n/2+2-$consistent, the points $(c_{t_1,j}, t_1), \ldots, (c_{t_{n/2+1},j}, t_{n/2+2})$, interpolate to a unique univariate degree $n/2+1$ polynomial (i.e. $f(x,j)$). This defines $(c_{1,j}, c_{2,j}, \ldots, c_{n,j})^T$, so the column j must be consistent with T. Since the jth column was an arbitrary column in S different from those in T, all such columns must be consistent with the rows defined be T. A symmetric argument shows that rows selected by S must be consistent with the columns selected by T. Therefore, both sets are consistent in that they define the same polynomials. Therefore, interpolation in $VSReveal_{\binom{n}{n/2+2}}$ will result in the same output. □

Lemma 3.10. *Let M be an $n \times n$ matrix with at most $n/2+1$ consistent indices. The probability that any $n/2+1$ randomly selected indices (without replacement) choose a set of $n/2+1$ consistent indices is no more than*

$$1/\binom{n}{n/2+1}$$

.

Proof. There can be at most 1 sets of size $n/2+1$ that is $n/2+1$ consistent in an $n \times n$ matrix. If there are fewer that $n/2+1$ consistent indices the probability of selecting a set of $n/2+1$ consistent indices is 0. Otherwise, we are left with the probability of choosing exactly one set of size $n/2+1$ from a set of n objects.

□

3.1.2 Modifying the SOA-secure Encryption Scheme to Support POKs

We will show a 2-round public-coin proof of knowledge of the encrypted bit based on any selective opening secure scheme. The protocol is neither zero-knowledge nor witness indistinguishable, but does maintain secrecy of the encrypted bit. First, we encrypt bits using the following protocol. Let $\Pi = (G, E, D)$ be a selective-opening attack secure scheme, such as the one we have described in Sec. 3. We construct a new encryption scheme $\tilde{\Pi} = (\tilde{G}, \tilde{E}, \tilde{D})$ to encode bits properly so we can give proofs of knowledge about them that keep the encrypted bit hidden. We define $\tilde{G} = G$, and given the algorithms for $\tilde{E}$ and $\tilde{D}$ below. For this section, all usage of Verifiable Secret Sharing uses the algorithms and definitions presented in Section 3.1.1.

```
$\tilde{E}(\mathrm{PK}, b, r)$
    $(s_1, ..., s_n) \leftarrow VSShare_{\binom{n}{n/2+2}}(b)$.
    Let $M$ be the $n \times n$ matrix
    representation of shares $(s_1, \ldots, s_n)$
    $c_{i,j} = E'(\mathrm{PK}, M_{i,j}, r_{i,j})$
    (These are bitwise encryptions of $M$)
    Output $\mathbf{C} = \{c_{i,j}\}_{1 \le i,j \le n}$.

$\tilde{D}(\mathrm{SK}, C)$
    $M = \{M_{i,j}\}_{1 \le i,j \le n} \leftarrow D'(\mathrm{SK}, \mathbf{C})$.
    Let $(s_1, \ldots, s_n)$ be the shares
    corresponding to matrix $M$.
    $T' = \{t | 1 \le t \le n$ share $s_t$
        is $n/2+2$ -consistent$\}$
    If $|T'| < n/2+2$ output $\perp$.
    Let $T \subseteq T'$ s.t. $|T| = n/2+2$.
    Output $VSReveal_{\binom{n}{n/2+2}}(s_{t_1}, ..., s_{t_{n/2+2}})_{t_i \in T}$.
```

3.1.3 Hidden Bit POK

Given a ciphertext $\mathbf{C} = \{c_{i,j}\}_{1 \le i,j \le n}$ output by our encryption algorithm $\tilde{E}$ and the random strings used to generate it, $\vec{r}$, we show how to perform a two-round proof of knowledge of the encrypted bit $\tilde{D}(\mathrm{SK}, \mathbf{C})$. P will prove that it has knowledge of the underlying shares of the verifiable secret-sharing scheme that have been encrypted, and thus the bit that has been encrypted. In order to do this, the verifier sends a random challenge of indices $T \subset \{1, \ldots, n\}$, where $|T| = n/2 - 1$. The encryptor then decommits to these encryptions by providing the random-bits used to encrypt each share of the bit. If each bit decommits successfully, and the result is $n/2-1$ valid shares to the VSS, then the verifier accepts.

```
Prover$(\mathrm{PK}, \mathbf{C} = \{c_{i,j}\}_{1\le i,j\le n}$                                   Verifier$(\mathrm{PK}, \mathbf{C} = \{\mathbf{c_{i,j}}\}_{\mathbf{1\le i,j\le n}})$
    $= \tilde{E}(\mathrm{PK}, b, r), M, r)$
Let $c_{i,j} = E(\mathrm{PK}, M_{i,j}, r_{i,j})$          $\xleftarrow{T}$                  $T \leftarrow \{S | S \subset \{1,...,n\} \wedge |S| = n/2-1\}$
                                   $\{M_{i,x}, r_{i,x}, M_{x,i}, r_{x,i}\}_{\substack{i\in T\\ 1\le x\le n}}$
                                                 $\longrightarrow$                  if $\exists i,j$ s.t. $c_{ij} \neq E(\mathrm{PK}, M_{i,j}, r_{i,j})$,
                                                                        output $\perp$.
                                                                    Output 1.
```

```
$Extractor(\mathbf{C}, \mathrm{PK}, U_1 = \{M_{i,x}, r_{i,x}, M_{x,i}, r_{x,i}\}_{\substack{i\in T_1\\ 1\le x\le n}}, U_2 = \{M_{i,x}, r_{i,x}, M_{x,i}, r_{x,i}\}_{\substack{i\in T_2\\ 1\le x\le n}})$
    Let $T = T_1 \cup T_2$, $U = U_1 \cup U_2$
    If $|T| < n/2$ output $\perp$.
    If $\exists i \in T, x \in \{1,\ldots,n\}$ s.t. $E(\mathrm{PK}, M_{i,x}, r_{i,x}) \neq c_{i,x}$ or $E(\mathrm{PK}, M_{x,i}, r_{x,i}) \neq c_{x,i}$ output $\perp$.
    For each $i \in T$ reconstruct its corresponding share $s_i$.
    Output $VSReveal_{\binom{n}{n/2+2}}(s_{r_1}, ..., s_{r_{n/2}})$,
where $r_1, \ldots, r_{n/2}$ are the smallest n/2 indices in $T$.
```

Completeness Follows by inspection.

Extractability (Soundness) Soundness follows from an extractor.

Theorem 3.11. *For all sufficiently large n, for all $d > 0$, for all $(SK, PK) \leftarrow \tilde{G}$, for all 'ciphertext' inputs C, and provers P', if $(P', V)(C = \{c_{i,j}\}_{1\le i,j\le n}, PK)$ accepts with probability $1/n^d$, then there exists a probabilistic polynomial time extractor that, with all but negligible probability, outputs a set of decommitments to all ciphertexts for a given set of indices $L = \{\ell_1, \cdots, \ell_{n/2+2}\} \subseteq [n]$ that constitute shares $S = \{s_{\ell_1}, ..., s_{\ell_{n/2+2}}\}$ such that $VSReveal_{\binom{n}{n/2+2}}(s_{\ell_1}, ..., s_{\ell_{n/2+2}}) = \tilde{D}(SK, C)$.*

Given the ability to rewind the prover-verifier protocol, we can extract the encrypted bit by recovering enough shares of the VSS scheme. We continue to execute the prover/verifier protocol until we get two distinct separate accepting proofs. It is a simple observation that except with exponentially small probability, we will succeed in $O(n^{d+1})$ rewinds. Let (T_1, U_1) and (T_2, U_2) be the flows in the first and second accepting proofs, respectively. By the security of the commitment scheme (Here we are using our encryption scheme as a simple commitment scheme), the probability that there is a ciphertext $c_{i,j}$ that is ever decommitted to in two distinct fashions is negligble.

We feed these inputs in to *Extractor*. We note that if there is not a valid encryption of a bit (fewer than $n/2+2$ committed and consistent shares), then the probability that the verifier outputs anything other than $\perp$ is bound to be less than $\frac{1}{\binom{n}{n/2+2}}$, by Lemma 3.10. This is exponentially small, and smaller than $1/n^d$ for any constant d for sufficiently large n.

Given the decommitments of the shares $\{s_i\}_{i \in T_i}$ for different randomly chosen set of indices T_1 and T_2, note these sets are not the same by selection, and therefore there is no chance that $\perp$ is output by the extractor. Next the extractor executes a $VSReveal_{\binom{n}{n/2+2}}$ command. However, this is not necessarily over the same shares as would be revealed in a legitimate decryption. We need to ensure that no matter which of the rewound and newly played legitimate traces we receive, we are going to reveal the same encrypted bit, with all but negligible probability. That is, we need to ensure that $VSReveal_{\binom{n}{n/2+2}}(s_{r_1}, ..., s_{r_{n/2}}) = VSReveal_{\binom{n}{n/2+2}}(s_1, ..., s_{n/2})$. This is the case, as shown in Lemma 3.9 because of the verifiable properties of the secret sharing scheme ensures that even in the case of a corrupted dealer (improper ciphertext encoding of shares) then all honest players will reveal the same value, with all but negligible probability. Therefore, with all but negligible probability we have that the extractor outputs the same value as $D(\mathrm{SK}, \vec{c})$.

Hidden Bit We show that no efficient cheating verifier can predict the bit b, when given $\vec{c} = \tilde{E}(\mathrm{PK}, b, r)$ as a theorem for which we are engaging in a POK.

Theorem 3.12. *For every P.P.T. adversary $A = (A_1, A_2)$, there exists a negligible function μ such that $\Pr[HB_A(1^k) = 1] \leq 1/2 + \mu(k)$, where the experiment HB is defined below:*

```
$HB_A(1^k)$
    $(PK, SK) \leftarrow \tilde{G}(1^k)$
    $b \in \{0,1\}$
    $\vec{c} = (\vec{c}_1, ..., \vec{c}_n) = \tilde{E}(PK, b)$
    where $\vec{c}_i = E(PK, s_i, r_i)$.
    $(T, \sigma) \leftarrow A_1(PK, \vec{c})$ where $T \subset \{1, \dots, n\}$, $|T| = n/2$.
    $b' \leftarrow A_2(\sigma, (s_i, r_i)_{i \in T})$
    Output $1$ iff $b = b'$
```

Proof. This follows directly from the IND-SO-SEC security of (G, E, D). Suppose an adversary $A = (A_1, A_2)$ breaks the hidden bit security of the protocol. That is for some c and infinitely many k: $\Pr[HB_A(1^k) = 1] \geq 1/2 + 1/k^c$. We use it to build an adversary $B = (B_1, B_2)$ and message selector M that breaks the IND-SO-SEC security of (G, E, D). The message selector M chooses a random bit b, and outputs $VSShare_{\binom{n}{n/2+2}}(b)$. The conditional message selector $M_{I, \vec{m}[I]}$ finds a random bi-variate polynomial of degree $n/2$ in each variable over the field F such that $f(0,0) \in \{0,1\}$ and for each $i \in I$, it holds that $((f(i,1), \dots, f(i,n)), (f(1,i), \dots, f(n,i))) = m_i$. By the information secrecy property of the VSS there are exactly the same number of such selections for the case $f(0,0) = 0$ and $f(0,0) = 1$. It is clear that such conditional message sampling can be done efficiently.

The adversary $B_1(\mathrm{PK}, E(\mathrm{PK}, m_1), \dots E(\mathrm{PK}, m_n))$ for the IND-SO-SEC experiment outputs $A_1(\mathrm{PK}, \vec{c} = (E(\mathrm{PK}, m_1), \dots E(\mathrm{PK}, m_n)))$. The adversary $B_2(\sigma, (m_i, r_i)_{i \in I}, \vec{m}^*)$ runs $VSReveal_{\binom{n}{n/2+2}}(\vec{m}^*) = b'$, it then executes $b \leftarrow A_2(\sigma, (m_i, r_i)_{i \in I})$ and outputs 1 iff $b = b'$.

Now consider $\Pr[B_{\Pi}^{\mathsf{Ind\text{-}SO\text{-}Real}}(1^k) = 1]$, this is a perfect simulation of $HB_A(1^k)$, and therefore is at least $1/2 + 1/k^c$. In contrast, consider $\Pr[B_{\Pi}^{\mathsf{Ind\text{-}SO\text{-}Ideal}}(1^k) = 1]$. In the case that $VSReveal_{\binom{n}{n/2+2}}(\vec{m}^*) = VSReveal_{\binom{n}{n/2+2}}(\vec{m})$, which occurs with probability exactly $1/2$, it is again a perfect simulation, and so outputs 1 with probability $1/2 + \epsilon$. In contrast, when $VSReveal_{\binom{n}{n/2+2}}(\vec{m}^*) \neq$

$VSReveal_{\binom{n}{n/2+2}}(\vec{m})$, then we know that B_2 outputs $VSReveal_{\binom{n}{n/2+2}}(\vec{m})$ with probability $1/2+\epsilon$, and so it must output 1 with probability $1-(1/2+\epsilon) = 1-\epsilon$. Therefore, $\Pr[B_{\Pi}^{\textsf{Ind-SO-Ideal}}(1^k) = 1] = (1/2)(1/2+\epsilon+1/2-\epsilon) = 1/2$. Therefore, $\Pr[B_{\Pi}^{\textsf{Ind-SO-Real}}(1^k) = 1] - \Pr[B_{\Pi}^{\textsf{Ind-SO-Ideal}}(1^k) = 1] \geq 1/k^c$, breaking IND-SO-SEC security.

4 Threshold FHE for the Integers

In this section, we first summarize secret sharing techniques that we use in the rest of this paper. We then describe the FHE scheme upon which we base our threshold FHE. In the remaining sections §4.4–§4.5, we describe our threshold scheme's key generation and decryption protocols.

4.1 Secret Sharing Primitives

Let F_q denote the finite field $\mathbb{Z}/q\mathbb{Z}$ with q being a prime. Let $[a]$ denote a secret-sharing of $a \in F_q$ and $a \leftarrow \textsc{reveal}([a])$ denote the execution of the reconstruction protocol for a in which the parties use their shares of $[a]$ as input to reconstruct the shared secret a. We refer to the simulator for $\textsc{reveal}$ protocol as $S_{\textsc{reveal}}([a]^I, a')$, which takes as input the shares of $[a]$ known by adversary $(denoted[a]^I)$, and reveal the value a'. This is done in a manner that is indistinguishable from the real world execution. Throughout the paper, in algorithmic descriptions we do not specify the field that secrets are shared in. *The field order must be large enough to handle the integers encountered in the fully homomorphic computations without causing wrap-around*, but is otherwise arbitrary. One can determine the field order needed by computing the maximum of the required orders for each FHE algorithm. Therefore in all secret sharing instances, we refer to the field order as an ℓ bit prime.

Let $[a]_B$ denote a shared value which is bit-decomposed. That is, every player holds a share of each bit of a. Also $[a]_{i,\ldots,j}$ denotes the bit-by-bit share of the the substring of a from the index i to the index j.

We assume that the secret sharing scheme is linear. Hence parties that hold the shares $[a]$ and $[b]$ can compute shares for $[a+b]$ and for $[ac]$ for a public constant c without interacting. We also assume that there is an unconditionally secure protocol $\textsc{mult}$ to compute $[ab]$ from the shares $[a]$ and $[b]$. Such a linear secret sharing scheme and a corresponding *constant round* multiplication protocol $\textsc{mult}$ that can be instantiated with protocols described in [Sha79] and [BOGW88]. Since we assume the existence of the multiplication protocol as an abstract primitive, we express the round complexity as the number of sequential multiplication calls that are necessary during the protocol. We also express the communication complexity as the number of the total multiplication calls during the protocol.

Some Known Primitives We first describe some known primitives from previous works on secret sharing. The protocols are used as building blocks to construct our shared key-generation and decryption protocols for the threshold decryption scheme. These protocols are all secure against static dishonest minority, and make use of atomic broadcast channels. We also describe slight modification to some of them, which are useful for our applications.

$\textsc{ran}_2()$ Damgård et al. [DFK$^+$06] give an n-party protocol $\textsc{ran}_2$ in which the players have no input, and receive as output shares of a uniformly distributed random bit $[a] \in \{0,1\}$. The protocol requires 2 sequential multiplication rounds. The simulator for this protocol is $S_{\text{RAN}_2}(b)$, where the bit b is the output the simulator is supposed to generate.

$\textsc{comp}([a]_B, [b]_B)$ [DFK+06], [NO07], and [Tof09] introduced a method to compare the bit-wise values $[a]_B$ and $[b]_B$ in a multiparty setup.The returned value is 1 if $a \geq b$, and 0 otherwise. The protocol runs in 8 rounds (two of which are preprocessing rounds), and by invoking $13\ell + 6\sqrt{\ell}$ multiplications, where ℓ is the bit length of the order of field in which a and b are shared. As mentioned in [Tof09], by adding another $\ell + \sqrt{\ell}$ multiplications, the comparison can be extended to return two bits that determine the case when $a = b$ as follows:

$$\textsc{comp}([a]_B, [b]_B) = \begin{cases} ([1],[0]) & \text{if } [a]_B > [b]_B \\ ([0],[0]) & \text{if } [a]_B = [b]_B \\ ([0],[1]) & \text{if } [a]_B < [b]_B \end{cases}$$

We use $\textsc{equal}([a]_B, [b]_B)$ to denote a protocol that returns 1 if the two outputs are equal, and 0 otherwise. This equality test is the same as running $\textsc{comp}([a]_B, [b]_B)$ and adding the two resulting bits, and then subtracting the sum from 1. To save space, we i) write $[c] \leftarrow \textsc{comp}([a]_B, [b]_B)$, to denote $c = 1$ if $a \leq b$, and 0 otherwise; and ii) whenever we need to find the maximum number in a set of k numbers, we write

$$[a_1], \ldots, [a_k] \leftarrow \textsc{comp}(x_1, \ldots, x_k),$$

where $[a_i] = 1$ if it is the maximum, and $[a_i] = 0$ otherwise. Computing this involves $\begin{pmatrix} k \\ 2 \end{pmatrix}$ parallel comparison of each pair of numbers. This modification to the $\textsc{comp}$ algorithm requires another $k-1$ rounds and $k(k-1)$ multiplications. Therefore, the overall round complexity is $7+k$ (two of which are for preprocessing) and $(k(k-1))(13\ell + 6\sqrt{\ell}) + k(k-1) = (k(k-1))(13\ell + 6\sqrt{\ell} + 1)$ multiplications. The simulator for this protocol is $S_{\text{COMP}}(([a]_B)^I, ([b]_B)^I, \mathsf{output})$, where output is the result of comparison simulated by simulator. Simulator for other variants of $\textsc{comp}$ and $\textsc{equal}$ are defined in a similar fashion.

$\textsc{bits}([a])$ It provide a method for taking $[a]$ and computing shares for each of the bits in the bit-wise representation of a through multi-party computation. The protocol returns $\ell = \lceil \log a \rceil$ shares of bits $([a_0], \ldots, [a_{\ell-1}])$, where $a = \sum_i a_i 2^i$. The protocol takes 23 rounds (7 of which are preprocessing rounds), and invokes $31\ell \log \ell + 71\ell + 30\sqrt{\ell}$ multiplications(ℓ is the bit length of the field order a is shared in). The simulator for this protocol is denoted $S_{\text{BITS}}(([a])^I, a_{\ell-1}, \ldots, a_0)$, where it gets the shares of $[a]$ that should be decomposed in to shares of a's bit-wise representation, as well as values $a_0, ..., a_{\ell-1}$ which each party would hold a share of at the end of the simulation. This algorithm was presented in [Tof09].

$\textsc{solved-bits}(k)$ This protocol returns shares of each bit of the bit-wise representation of a value chosen uniformly at random in the range $[0, k]$. The protocol takes seven rounds (two of which are preprocessing rounds), and $52\ell + 24\sqrt{\ell}$ multiplications. The simulator for this protocol is $S_{\text{SOLVEDBITS}}(a)$, where a is the simulated random output. This algorithm was presented in [Tof09].

$\mathsf{power}([x], k)$ This algorithm returns shares of x^k in the case that x is an invertible field element. The simulator for this protocol is $\mathsf{powerSim}^*(([x])^I, k, y)$. The simulator should give the shares of y to represent the shares of x^k to all parties during the simulation. This protocol is a simple modification of a protocol based on the works of [BIB89] and [DFK$^+$06] which given as input shares $[x_1], ..., [x_k]$ returns in constant rounds shares for all of the values $1 \leq i < j \leq k$, $\prod_{k=i}^{j} x_k$. The protocols takes 5 rounds and incurs $5\ell + k - 1$ multiplications.

MOD$([x], m)$ This protocol computes shares of $x \mod m$, where m is a public value. The simulator for this protocol is $S_{\mathbf{MOD}}([x], m, a)$, where the simulator produces shares of a, with the intention that $[x] = a \mod m$. The protocol takes up to 40 rounds, and $31\ell^2 \log \ell + 31\ell \log \ell + 84\ell^2 + 71\ell + 36\ell\sqrt{\ell} + 30\sqrt{\ell}$ multiplications. This protocol is a natural augmentation of a protocol presented in [DFK$^+$06], that results from using techniques in [Tof09] to improve efficiency.

4.2 Approximate-GCD scheme

We start by summarizing the FHE scheme based on the Approximate-GCD problem described by [vDGHV10]. The scheme relies on the boot-strapping principle and is based on a somewhat homomorphic scheme parameterized by the following variables:

γ is the bit-length of the integers in all the somewhat homomorphic scheme's public key,

τ is the number of integers in the somewhat homomorphic scheme's public key,

η is the bit-length of the secret key in somewhat homomorphic scheme (which is the hidden approximate-gcd of the integers in the public-key),

ρ is the bit-length of the noise.

For the security parameter λ, vanDijk [vDGHV10] suggests the following relationships:

- $\rho = \omega(\log \lambda)$, to protect against brute-force attacks on the noise;
- $\eta \geq \rho \cdot \Theta(\lambda \log^2 \lambda)$, in order to support homomorphism for deep enough circuits to evaluate the "squashed decryption circuit";
- $\gamma = \omega(\eta^2 \log \lambda)$, to thwart various lattice-based attacks on the underlying approximate-gcd problem;
- $\tau \geq \gamma + \omega(\log \lambda)$, in order to use the leftover hash lemma in the reduction to approximate gcd.
- $\rho' = \rho + \omega(\log \lambda)$,

Now the parameters are set as following: $\rho = \lambda$, $\rho' = 2\lambda$, $\eta = \tilde{O}(\lambda^2)$, $\gamma = \tilde{O}(\lambda^5)$ and $\tau = \gamma + \lambda$.

For $z \in \mathbb{R}$, define $r_p(z) = z - \lfloor z/p \rceil \cdot p$, i.e, it is the remainder in the range $(-p/2, p/2)$. For a specific (ρ-bit) odd positive integer p, let the distribution $D_{\gamma,\rho}(p)$ over γ bit integers be:

$$D_{\gamma,\rho}(p) = \{\text{choose } q \leftarrow Z \cap [0, 2^\gamma/p), r \leftarrow Z \cap (-2^\rho, 2^\rho) : \text{output } x = pq + r\}$$

4.3 Public Key Homomorphic Encryption Scheme

The homomorphic encryption scheme $\varepsilon = (\mathbf{Gen}', \mathbf{Enc}', \mathbf{Eval}', \mathbf{Dec}')$ works as follows:

$\mathbf{Gen}'(\lambda)$ The secret key is an odd η-bit integer: $p \leftarrow (2Z+1) \cap [2^{\eta-1}, 2^{\eta})$.

For the public key, sample $x_i \leftarrow D_{\gamma,\rho}(p)$ for $i = 0, \ldots, \tau$. Relabel the vector $\vec{x}$ so that x_0 is the largest integer. Restart unless x_0 is odd and $r_p(x_0)$ is even. The public key is $pk = \langle x_0, x_1, \ldots, x_\tau \rangle$.

$\mathbf{Enc}'(pk, m \in \{0,1\})$ Choose a random subset $S \subseteq \{1, 2, ..., \tau\}$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$. Output $c^* \leftarrow [m + 2r + 2\sum_{i \in S} x_i]_{x_0}$.

$\mathbf{Eval}'(pk, C_\varepsilon, c_1, ..., c_t)$ Given an (arithmetic) admissible circuit C_ε with t inputs, and t ciphertexts c_i, apply the (integer) addition and multiplication gates of C_ε to the ciphertexts, performing all the operations over the integers, and return the resulting integer.

If $f(x_1, \ldots, x_t)$ is the multivariate polynomial representation of the circuit C_ε, and f is of degree d, then we say that C_ε is admissible if:

$$d \leq \frac{\eta - 4 - \log|f|}{\rho' + 2}$$

where $|f|$ is the l_1 norm of the coefficient vector of f.

$\mathbf{Dec}'(sk, c)$ Output $m' \leftarrow (c \mod p) \mod 2$.

The scheme can be transformed into a fully homomorphic one by applying the bootstrapping transformations described in [vDGHV10]. In particular, the decryption depth of the circuit must be squashed by adding extra information to the public key and modifying the encryption procedure. Towards this goal, set the additional parameters $\kappa = \gamma\eta/\rho'$, $\Theta = \omega(\kappa \log \lambda)$, and $\theta = \lambda$. Define the scheme $\Pi = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Eval}, \mathbf{Dec})$ as follows:

$\mathbf{Gen}(\lambda)$ Generate $sk^* = p$ and pk^* as before. Set $x_p \leftarrow \lfloor 2^\kappa / p \rceil$. Choose at random a Θ-bit vector $\vec{s} = \langle s_1, \ldots, s_\Theta \rangle$ with Hamming weight θ, and let $S = \{i : s_i = 1\}$. Choose at random integers $u_i \in Z \cap [0, 2^{\kappa+1})$ for $i = 1, \ldots, \Theta$, subject to the condition that $\sum_{i \in S} u_i = x_p \pmod{2^{\kappa+1}}$. Set $y_i = u_i / 2^\kappa$ (it is a real number with $\lceil \log \theta + 3 \rceil$ bits of precision) and $\vec{y} = \langle y_1, \ldots, y_\Theta \rangle$. Output the secret key $\mathrm{SK} = (\vec{s})$ and public key $\mathrm{PK} = (pk^*, \vec{y})$. (Notice that p is no longer needed in the secret key.)

$\mathbf{Enc}(m, \mathbf{PK})$ Generate a ciphertext c^* as before in $\mathbf{Enc}'$ (i.e., an integer). Then for $i \in 1, ..., \Theta$, set $z_i \leftarrow [c^* \cdot y_i]_2$, keeping only $\lceil \log \theta \rceil + 3$ bits of precision after the binary point for each z_i. Output both c^* and $\vec{z} = \langle z_1, ..., z_\Theta \rangle$.

$\mathbf{Dec}(c = \langle c^*, \vec{z} \rangle, \mathbf{SK})$ Output $m' \leftarrow [c^* - \lfloor \sum_i s_i z_i \rceil]_2$.

Theorem 4.1 (Implicit from [vDGHV10]). *Fix the parameters $(\rho, \rho', \eta, \gamma, \tau)$ and (κ, Θ, θ) as noted above. Under the assumption that the (ρ, η, τ) approximate-GCD problem and the (θ, Θ)-sparse subset sum problem are hard, Π is a semantically-secure compact bootstrappable somewhat homomorphic encryption scheme.*

4.4 Sharing the Public and Secret Key

In this section, we describe a constant-round, n-party protocol to generate both a public key and shares of the secret key for the fully homomorphic encryption scheme Π. Our schemes rely on the secret sharing sub-protocols described in section 2.

Recall that the secret-key for Π consists of a Θ-bit vector $\vec{s}$ with Hamming weight θ. In fact, we note that instead of θ, it suffices to select a vector with Hamming weight in the interval $\theta \pm \theta/4$. This observation allows us to pick a SK by independently flipping coins that are 1 with probability θ/Θ.

Generating the public key is more complicated. The public key consists of the vectors $\vec{x}$ and $\vec{u}$. There are 3 steps in generating the public key. In order to generate $\vec{u}$, we first compute shares of p which is an odd integer in the interval $[2^{\eta-1}, 2^{\eta})$. Second, we compute $x_p = \lfloor 2^{\kappa}/p \rceil$. Using $\vec{s}$ and x_p, we compute the vector $\vec{u} = \sum_i s_i \cdot u_i \mod 2^{\kappa+1}$. Third, using bits of $1/p$ computed in previous steps, we generate the x_i's. In the next sections, we provide more details on each of these steps.

4.4.1 Producing the SK $\vec{s}$

The secret key for the squashed scheme consists of a random Θ-bit vector $\vec{s} = (s_1, \ldots, s_\Theta)$ with Hamming weight θ. We argue that setting the Hamming weight of $\vec{s}$ to be any value in the range $\theta \pm \theta/4$ does not affect the security or correctness of the scheme. To verify this, note that the sparse subset-sum problem is assumed to be hard for $\theta = \Theta^{\epsilon}$ for $0 < \epsilon < 1$; our change does not violate this condition. Also, our new range of settings for θ does not increase the total degree of the decryption circuit by more than a factor of 2 and thus the condition that **Dec** is admissible is maintained (and thus the scheme is bootstrappable. See the computation on p.18 [vDGHV10].) Our approach for producing $\vec{s}$ is to securely generate a random number r_i in the range $[0, \Theta]$ for each s_i and setting $s_i = 1$ if $r_i \leq \theta$ and 0 otherwise.

Claim 1. *If each s_i is set to 1 with probability θ/Θ, then*

$$\Pr\left[\left|\sum_i s_i - \theta\right| > \theta/4\right] \leq 2^{-O(\lambda)}$$

Proof. Via the Chernoff bound. □

We also assume that the circular threshold security of the framework still holds with this modification. We believe that any natural proof in showing this modification still results in circular security, would modify the original circular security argument for the base system. However, remember that the circular security of the original scheme is assumed, and therefore we cannot modify such a proof.

We set the parameter Θ to be a power of two to facilitate generating secret random elements smaller than Θ. In this case, generating secret random numbers smaller than Θ only requires concatenation of $\log \Theta$ secret random bits without any secret comparison. However in the general case, we would call the COMP protocol, but this is a relatively expensive operation. Therefore the algorithm to compute $\vec{s}$ is as follows: For each s_i, the players produce shares of $\log \Theta$ random bits in step 1 (notice that the concatenation of these bits would be guaranteed to be in the interval $[0, \Theta]$). After local computation in step 2, the players securely compare the result against θ to generate shares of s_i (i.e., s_i would be 1 with probability θ/Θ, and 0 otherwise).

Protocol 3 . $[\vec{s}], \theta' \leftarrow \text{ComputeS}(\theta, \Theta)$

```
1: For $i = 1, \dots, \Theta$ and $j = 1, \dots, \log\Theta$, the players run $[a_{i,j}] \leftarrow$ RAN2() to generate shares of random bits.
2: For $i = 1, \dots, \Theta$, each player locally computes $[a_i] \leftarrow \sum_j [a_{i,j}] \cdot 2^i$.
3: For $i = 1, \dots, \Theta$, the players run $\Theta$ parallel executions of protocol $[s_i] \leftarrow$ COMP($[a_i], \theta$). At the end of this protocol, each player has a share of bit $[s_i]$.
4: Players locally compute $[\theta'] \leftarrow \sum_i [s_i]$.
5: Players run the protocol $\theta' \leftarrow$ REVEAL($[\theta']$) to reconstruct the value $\theta'$.
6: Output $[\vec{s}], \theta'$
```

Complexity Analysis The algorithm produces $\Theta \log \Theta$ random bits in parallel, and performs another Θ comparisons, that again can be done in parallel. Therefore it needs $2\Theta\log\Theta + \Theta(13\ell + 6\sqrt{\ell})$ invocations of the multiplication protocol and 7 rounds of interaction.
Simulation. The simulator $S_S(\vec{s'})$ of ComputeS works as follows:

1. Run the sub-simulator $S_{\text{RAN}_2}(0)$ for each call of RAN2 protocol and obtain the adversary's share. Next, do the local computation in step 2 and obtain $[\vec{a}]^I$.
2. For all $i \in [1, \Theta]$ run the sub-simulator $S_{\text{COMP}}([a_i]^I, \theta, s'_i)$, and obtain the adversary's share $[\vec{s'}]^I$.
3. Locally compute $[\theta']^I = \sum_i [s'_i]^I$.
4. Run $S_{\text{REVEAL}}([\theta']^I, \sum_i s'_i)$.
5. Output $[\vec{s'}]^I$ and θ'.

4.4.2 Computing p

The secret key p for the "somewhat homomorphic encryption scheme" is an odd η-bit integer. To sample p, we notice that the ${p_0}^{\text{th}}$ and ${p_{\eta-1}}^{\text{th}}$ bits should be 1 whereas the rest of the bits $p_1, \dots, p_{\eta-2}$ should be generated by having the players execute RAN2(). Therefore, the secret key would be $2^{\eta-1} + \sum_{i=1}^{\eta-2} p_i 2^i + 1$ (which can be computed locally by players from shares of the bits). At the end, each player holds a share of p. The computation complexity involves $2(\eta - 2)$ multiplication invocations and 2 rounds of interaciton. The simulator for this subprotocol, named $S_{\text{MOD}}(p')$, is defined by calling $S_{\text{RAN}_2}(p'_i)$ for $i \in [1, \eta - 1]$. The simulator outputs $[p']^I$.

4.4.3 Computing $\lfloor \frac{2^\kappa}{p} \rceil$

In [KLML05], the authors present a method for two honest-but-curious parties to compute the average of their inputs. We extend their technique to allow multiple parties who hold shares of p to compute shares of $1/p$, and address the malicious model.We generalize [KLML05]'s approach to calculate $\lfloor 2^\kappa / p \rceil$ by computing the first κ bits of $1/p$ and then rounding.

Recall that p is subject to the constraint $2^{\eta-1} \leq p < 2^{\eta}$; set $\epsilon \in [0, 1/2]$ such that $p = 2^{\eta}(1-\epsilon)$. Thus:

$$p^{-1} = 2^{-\eta} \cdot \frac{1}{(1-\epsilon)} = 2^{-\eta} \sum_{i=0}^{\infty} \epsilon^i = 2^{-\eta} \left(\sum_{i=0}^{d} \epsilon^i \right) + 2^{-\eta} R_d$$

where $0 \leq R_d < 2^{-d}$. Multiplying both sides by $2^{\eta(d+1)}$ yields

$$2^{\eta(d+1)} p^{-1} = \left(\sum_{i=0}^{d} (2^{\eta}\epsilon)^i 2^{\eta(d-i)} \right) + \left(2^{\eta d} R_d \right) \tag{1}$$

Notice that $2^{\eta}\epsilon$ is an integer (since $p = 2^{\eta}(1-\epsilon)$ is an integer).

Let Z denote the first summand in 1 (i.e., $\sum_i (2^{\eta}\epsilon)^i 2^{\eta(d-i)}$). Having shares of p, players compute $2^{\eta}\epsilon$ collaboratively using the formula $2^{\eta}\epsilon = 2^{\eta} - p$. Holding shares of $2^{\eta}\epsilon$ and using the protocol power for exponentiation, the players can now compute shares of Z.

Because the exact integer value of Z is desirable, we need to choose the field Z_l used in the secret sharing scheme to be large enough to ensure $[Z]_l = Z$. In order to determine the bit-length of the field we first determine the maximum value $Z = \sum_{i=0}^{d} (2^{\eta}\epsilon)^i 2^{\eta}(d-i)$ can take. By 1 we have:

$$\sum_{i=0}^{d} (2^{\eta}\epsilon)^i 2^{\eta(d-i)} = 2^{\eta(d+1)} p^{-1} - 2^{\eta d} R_d$$

Our constraints ensure that $2^{\eta d} < 2^{\eta(d+1)} p^{-1} \leq 2^{\eta d+1}$ which immediately implies $Z \leq 2^{\eta d+1}$. The constraints $0 \leq R_d < 2^{-d}$ imply that $\log(2^{\eta d} R_d) < \eta d - d$. But earlier we showed:

$$\eta d < \log(2^{\eta(d+1)} p^{-1}) \leq \eta d + 1$$

These two facts ensure that the error term $2^{\eta d} R_d$ will only change the least significant $\eta d - d$ bits in $2^{\eta(d+1)} p^{-1}$. The difference of the two bit lengths, $\eta d - (\eta d - d) = d$, is the number of bits that the error term does not change (assuming a carry will not happen). For our purposes, it suffices to compute the first κ bits of $1/p$ to yield $2^{\kappa}/p$. Therefore, we set $d = \kappa$.

Note that $[Z]_{\kappa d \ldots \kappa d - \kappa}$ or $[Z]_{\kappa d \ldots \kappa d - \kappa} + 1$ is the integer value of $\lfloor \frac{2^{\kappa}}{p} \rfloor$. The reason for potentially adding one to the value is that in our calculation of Z, we do not include the summand $2^{\kappa d} R_d$. This summand might have propagated a carry that changes the value of $[Z]_{\kappa d .. \kappa d - \kappa}$. Further, $\lfloor \frac{2^{\kappa}}{p} \rceil$ must be either $\lfloor \frac{2^{\kappa}}{p} \rfloor$ or $\lfloor \frac{2^{\kappa}}{p} \rfloor + 1$. Therefore, x_p is either $x_p = [Z]_{\kappa d \ldots \kappa d - \kappa}$, $x_p = [Z]_{\kappa d \ldots \kappa d - \kappa} + 1$, or $x_p = [Z]_{\kappa d \ldots \kappa d - \kappa} + 2$. We can determine which case holds by multiplying each of the candidates by p and testing which result has the smallest distance to 2^{κ}. To perform this test, notice that if we took any of the three computed values and subtracted 2^{κ} to determine proximity (as is done on line 10 of Protocol 4.4.3 (ComputeXP)) the result might be negative. Since we are in finite field this causes problems, as it results in a large positive number, destroying our notion of closeness. For example, imagine subtracting 2^{κ} from the three values $2^{\kappa} - 1$, $2^{\kappa} + p - 1$, $2^{\kappa} + 2p - 1$. Then the subtraction result would be $-1 = \ell - 1$, $p - 1$, and $2p - 1$. The value $-1 = \ell - 1$ corresponds to the correct choice, but a direct smallest magnitude comparison would be pick the second value, $p - 1$. To overcome this problem, we square the result of the subtraction (line 11 of Protocol 4.4.3 (ComputeXP) below) so that all values are positive and the comparison has the desirable result.

There is no fear of modular reduct, as it is easy to observe that any of subtracted results lie in $[-3p+1, 3p-1]$, and squaring these values results in value smaller than the modular reduction of ℓ.

In the following protocol we formalize the above reasoning. We also set x_γ as the first γ bits of $[Z]_{\eta\kappa..\eta\kappa-\kappa}$. It is an output of the function. This value is needed later for generating the public key $\langle x_0, \ldots, x_\tau \rangle$.

Protocol 4. $[x_p] \leftarrow \text{ComputeXP}([p], \kappa, \eta)$

```
1: Locally compute shares $[e] \leftarrow [2^\eta] - [p]$.
2: Execute the protocol $([e], [e^2], \ldots, [e^\kappa]) \leftarrow \mathsf{power}([e], \kappa)$ to generate shares of $e^i$ for $i \in 1, \ldots \kappa$.
3: Locally compute shares $[Z] \leftarrow \sum_{i=0}^{\kappa} [e_i] 2^{\eta(\kappa-i)}$.
4: Execute bit-decomposition protocol to generate shares $[Z]_B \leftarrow \textsc{bits}([Z])$.
5: Locally generate a share of $[Z]_{\eta\kappa..\eta\kappa-\kappa}$ by using the bit-decomposed shares $[Z]_B$ and the constants $2^j$ for $j \in [0..\kappa-1]$.
6: Locally set $[x_\gamma]_B = [Z]_{\eta\kappa..\eta\kappa-\gamma}$.
7: Run a trivial protocol to generate shares of the constants $[0]$, $[1]$, $[2]$, and $[2^\kappa]$.
8: Locally compute $[x_i] \leftarrow [Z]_{\eta\kappa..\eta\kappa-\kappa} + [i]$.
9: For $i = 0..2$, run protocol $[x'_i] \leftarrow \textsc{mult}([x_i], [p])$.
10: For $i = 0..2$, locally compute $[x''_i] \leftarrow [2^\kappa] - [x'_i]$.
11: For $i = 0..2$, run protocol $[x''_i] \leftarrow \textsc{mult}([x''_i], [x''_i])$.
12: For $i = 0..2$, run protocol $[x''_i]_B \leftarrow \textsc{bits}([x''_i])$.
13: Run protocol $[a_0], [a_1], [a_2] \leftarrow \textsc{comp}([x''_0]_B, [x''_1]_B, [x''_2]_B)$.
14: Run protocol $[x_p] \leftarrow [a_0][x_0] + [a_1][x_1] + [a_2][x_2]$.
15: Output $[x_p]$, $[x_\gamma]_B$.
```

Complexity Analysis In this protocol, the size of the field for secret sharing should be at least $2^{\eta\kappa+1}$. The above protocol invokes the power subprotocol once in computing $[e]$ and κ times on line 2; the $\textsc{bits}$ subprotocol once on line 4 (5 out of 7 preprocessing rounds can be run parallel with $\textsc{mult}^*$) and three times on lines 12 (7 preprocessing rounds can be run parallel with previous lines); and the $\textsc{comp}$ subprotocol once with 3 inputs on line 13. Additionally, 3 multiplications are invoked on each of the lines 9, 11, and 14. Therefore these all multiplication invocation numbers adds up to: $124\ell \log \ell + 328\ell + 138\sqrt{\ell} + 11$. Round complexity analysis is as follows: line 2 takes 5 rounds; lines 4 and 12 take 21 rounds each (we can run the 2 preprocessing rounds in advance); line 13 takes 7 rounds; and lines 9, 11, and line 14 each take 1 round. The result is a total of $5 + 18 + 16 + 7 + 1 + 1 + 1 = 49$ rounds. Note that $[x_\gamma]_B$ is computed by the end of 23thround and can be returned at that point.

Simulation. The simulator for protocol 4, $S_{\mathbf{XP}}([p]^I, x_p, x_\gamma)$, is described as follows:

1. The first step of the protocol is a local computation.
2. For step 2, the simulator calls $S^*_{\mathbf{MULT}}(([e])^I, d, 0)$. At the end of this step, each player holds shares of 0 for each bit of e and the simulator learns all the shares held by the adversary.

3. For Step 3 is local computation. The simulator maintains knowledge of the shares of $[Z]^I$.

4. For Step 4, the simulator calls $S_{\mathbf{BITS}}([Z]^I, x_\gamma)$ and learns $[Z]^I_B$.

5. For Steps 5 to 13, and for the first two multiplications in step 14, the simulator calls the related sub-simulator for each step, with the adversary's input of shares and the output of zero for that sub-simulator. Trivially each step's adversary input share can be obtained from the shares it obtained from previous step.

6. For the last multiplication in step 14, the simulator calls $S_{\mathbf{MULT}}([a_2]^I, [x_2]^I, x_p)$. Then the simulator adds up the shares from the three terms as their share for $[x_p]^I$.

7. Output $[x_p]^I$ and $[x_\gamma]^I_B$.

4.4.4 Producing $\vec{y}$

The $\vec{s}$ vector computed Section 4.4.1 is used to select the hidden subset-sum in the vector $\vec{y}$ that sums up to $\approx 1/p \mod 2$. In [vDGHV10], the vector $\vec{y}$ is generated as follows:

1. Sample integers $u_i \in Z \cap [0, 2^{\kappa+1})$, $i \in [1, \Theta]$, such that $\sum_i u_i \cdot s_i = x_p \mod 2^{\kappa+1}$

2. For $i = 1, \ldots, \Theta$, set $y_i = u_i / 2^\kappa$ (using κ bits of precision)

Since $\vec{u}$ reveals all information in $\vec{y}$ and nothing more, it suffices to compute and reveal $\vec{u}$. To compute $\vec{u}$, we generate random numbers in the interval $[0, 2^{\kappa+1})$ for each u_i in lines $1-2$ (no information is revealed since the generated values are random integers). Next, we need modify the vector $\vec{u}$ to satisfy the constraint:

$$\sum_i u_i \cdot s_i = x_p \bmod 2^{\kappa+1}.$$

We do so by selecting a value k at random s.t. $s_k = 1$, and set $u_k = (x_p - \sum_{i \neq k} u_i \cdot s_i)$. However, in doing so we will reveal the value k to all parties, and in so doing we release part of the secret-key. It is relatively easy to argue, and is done in Claim 2, that while this slightly reduces security, a secure scheme remains secure.

To perform the calculations just described, we choose a random index $k \in [0..\theta]$ in step 3 and then find the index of the k^{th} 1 in the vector $\vec{s}$ in line 2. Denote this index as k^*. Computing this index is done as follows, each player locally computes prefix sums $\sum_{j=0}^{i} [s_i]$ for each value i, and tests for equality with $[k]$. The results is a set of shares $[s'_i]$ which are all zero except at the indices between the k^{th} and the $k+1^{\text{th}}$ occurrence of 1 in $\vec{s}$. To isolate the occurrence corresponding to the index, we multiply $[s'_i]$ with $[s']$. Finally, we reveal this index in the form of shares $[s''_i]$. Notice that $s''_{k^*} = 1$ and all other positions will be 0. The above procedure is illustrated in Table 1. Finally, we modify the values of u_{k^*} to satisfy the constraint: We replace the u_k with $2^{\kappa+1} - 1$ (to make sure the result of subtraction in step 9 would not be negative), compute $dif = x_p - \sum_i u_i \cdot s_i \mod 2^{\kappa+1}$, and reveal the result. The users, then, can locally replace u_k with $u_k - dif \mod 2^{\kappa+1}$ in step 11 which guarantees that $\sum_i u_i \cdot s_i = x_p \mod 2^{\kappa+1}$.

We now show that this modification of Π neither weakens its indistinguishability security nor its circular threshold security.

Table 1: Example of producing $\vec{s''}$ for $\vec{s} = (0,1,0,1,0,0,1,1,0,0)$ and $k = 2$

i	0	1	2	3	4	5	6	7	8	9
s_i	0	1	0	1	0	0	1	1	0	0
$\hat{s}_i = \sum_{j=0}^{i}[s_j]$	0	1	1	2	2	2	3	4	4	4
$s'_i = \text{EQUAL}([\hat{s}_i], [2])$	0	0	0	1	1	1	0	0	0	0
$s''_i = \text{MULT}([s'_i] \cdot [s_i])$	0	0	0	1	0	0	0	0	0	0

Claim 2. *Let Π' be the encryption scheme that results from the modifications to the keygen described above. An adversary A' that breaks the indistinguishability threshold security of Π' with advantage ϵ can be used to construct an adversary A that breaks Π with advantage $p(\epsilon)$ for some polynomial p.*

Proof. (Sketch). The adversary A chooses a random index k. With non-negligible probability, k is the index of a 1 in the vector $\vec{s}$. Then, A calls A' with the input k. A has θ'/Θ chance of guessing k correctly (non-negligible), and A' has ϵ chance of breaking security. therefore A can break the security with probability $\epsilon' = \epsilon\theta'/\Theta$. □

Claim 3. *Let Π' be the encryption scheme that results from the modifications to the keygen described above. An adversary A' that breaks the circular threshold security of Π' with advantage ϵ can be used to construct an adversary A that breaks the circular security of Π with advantage $p(\epsilon)$ for some polynomial p.*

Proof. The proof is similar to the proof of claim 2. □

Protocol 5. $[\vec{y}] \leftarrow \text{ComputeY}([x_p], [\vec{s}], \theta')$,

```
1: For $i = 1..\Theta$, and for $j = 0..\kappa$ run protocol $[u_{i,j}] \leftarrow \text{RAN}_2()$ in parallel.
2: For $i = 1..\Theta$ locally compute $u_i \leftarrow \sum_j [u_{i,j}]2^j$.
3: Run protocol $[k] \leftarrow \text{SOLVED-BITS}(\theta')$.
4: For $i = 1..\Theta$ run protocol $[s'_i] \leftarrow \text{EQUAL}(\sum_i [s_i], [k])$ in parallel.
5: For $i = 1..\Theta$ run protocol $[s''_i] \leftarrow \text{MULT}([s_i], [s'_i])$ in parallel.
6: For $i = 1..\Theta$: run protocol $s''_i \leftarrow \text{REVEAL}([s''_i])$ in parallel.
7: $s'' \leftarrow i$ such that $s''_i = 1$.
8: Locally compute $u_{s''} \leftarrow 2^{\kappa+1}$.
9: $[dif] \leftarrow (\sum_i u_i \cdot [s_i] - [x_p] \mod 2^{\kappa+1})$.
10: Run protocol $dif \leftarrow \text{REVEAL}[dif]$.
11: Locally compute $u_{s''} \leftarrow -dif$.
12: Output $\vec{u}/2^{\kappa}$.
```

Complexity Analysis. ComputeY calls RAN$_2$ $\Theta(\kappa + 1)$ times in line 1, and SOLVED-BITS(θ) once in line 3. Subprotocols EQUAL and MULT subprotocols are each called Θ times in lines 4 and 5. Next, $\mod 2^{\kappa+1}$ is computed in line 9, which is done by a call to the BITS subprotocol on the input and taking only the least significant $\kappa + 1$ bits as the result. Therefore, the round and multiplication complexity of this step is the same as the BITS subprotocol. Totaling, this protocol requires $31\ell \log \ell + (13\Theta + 123)\ell + (6\Theta + 54)\sqrt{\ell} + \Theta + 2\Theta(\kappa + 1)$ rounds of interaction

and $2+6+5+1+16=30$ multiplications. The largest value computed is upper bounded by $\theta(2^{\kappa+1})$ in line 9, therefore ℓ needs to have a bit-length of at least $|\theta|(\kappa+1)$.
Simulation. The simulator $S_{\mathrm{Y}}([x_p]^I, [\vec{s}]^I, \vec{y'})$ acts as follows:

1. Let $\vec{u'} = \vec{y'} \cdot 2^{\kappa}$,
2. The simulator picks a random k' as the index of $\vec{u}$ that should be revealed to the adversary. Then for the first step, the simulator calls the simulator $S\textsc{Ran}_2(u_{i,j})$ for each bit of each of the u_i's except for u'_k and learns $[u_{i,j}]^I$. For $i=k'$, the simulator picks random r_j's, calls $S\textsc{Ran}_2(r_j)$, and again learns $[u'_k]^I$. Then the simulator locally computes $[u_i]^I$,
3. Fore step 3, the simulator calls $S_{\mathbf{SOLVEDBITS}}(\theta', 0)$ and learns $[k]^I$,
4. For step 4, the simulator calls $S_{\mathbf{EQUAL}}(\sum_i [s_i]^I, [k]^I, 0)$ and learns $[s'_i]^I$,
5. For step 5, the simulator calls $S_{\mathbf{MULT}}([s_i]^I, [s'_i]^I, 0)$ and learns $[s''_i]^I$,
6. For all $i \in [1, \Theta]$ except for k', the simulator calls the simulator $S_{\mathbf{REVEAL}}([s''_i]^I, 0)$. Then for $i=k'$, the simulator calls $S_{\mathbf{REVEAL}}([s''_i]^I, 1)$,
7. Set $s'' = i$ s.t. $s''_i = 1$
8. The simulator follows the algorithm in step 8 and computes $dif' = -u'_{s''} \mod 2^{\kappa+1}$,
9. The simulator calls $S_{\mathbf{BITS}}((\sum_i u_i [s_i]^I - [x_p]^I), 0)$ for step 9 to get $[dif]^I$,
10. The simulator calls $S_{\mathbf{REVEAL}}([dif]^I, dif')$,
11. Output $\vec{y'}$.

4.4.5 Computing $\langle x_0, \ldots, x_\tau \rangle$

Recall from the original public-key generation algorithm that we need to sample $x_i \leftarrow D_{\gamma,\rho}(p)$ for $i = 0, \ldots, \tau$. Intuitively, these x_i represent random encryptions of 0 that get added to our base encryption in the homomorphic scheme. Further, recall that

$$D_{\gamma,\rho}(p) = \{\text{choose } q \leftarrow Z \cap [0, 2^{\gamma}/p), r \leftarrow Z \cap (-2^{\rho}, 2^{\rho}) : \text{ output } x \leftarrow pq + r\}.$$

After sampling, the list should be relabeled so that x_0 is the largest. The key-generation process requires that the process is restarted if either x_0 is even or $x_0 - \lfloor x_0/p \rceil \cdot p$ is odd. Since $x_0 = pq + r$ is generated as directed for some random q and r and since p is an odd number, the requirement that x_0 is odd can be checked by inspecting the least significant bits of the q and r: If $q_0 + r_0 = 1$, then x_0 satisfies the first condition.

To check the second condition, that $x_0 - \lfloor x_0/p \rceil \cdot p$ is an odd number, we observe that because of the constraints $-2^{\rho} < r < 2^{\rho}$ and $2^{\eta-1} \leq p < 2^{\eta}$, it follows that

$$-2^{\rho-\eta+1} < r/p < 2^{\rho-\eta+1}$$

Since $\rho = \lambda$ and $\eta = \tilde{O}(\lambda^2)$, therefore for all sufficiently large λ (if $\eta = \lambda^2$, then for $\lambda > 2$), $\lfloor r/p \rceil = 0$ and as a result r can be ignored. That is $\lfloor x_0/q \rceil = \lfloor pq + r/q \rceil = q + \lfloor r/q \rceil = q$. So

$x_0 - \lfloor x_0/p \rceil \cdot p = x_0 - q \cdot p$. Because x_0 and p are both odd, q must be odd to make the term $x_0 - \lfloor x_0/p \rceil \cdot p$ even. These constraints imply that for x_0 to be odd and $x_0 - \lfloor x_0/p \rceil \cdot p$ to be even, then q must be even and r must be odd.

To sample $q \in [0, 2^\gamma/p)$, we first compute $\lfloor 2^\gamma/p \rfloor$. The bit decomposition of $\lfloor 2^\kappa/p \rfloor$ (or potentially the bit decomposition of $\lfloor 2^\kappa/p \rfloor - 1$, but it does not matter since it makes negligible difference) from the protocol ComputeXP as $[x_\gamma]_B$ can be used to compute $\lfloor 2^\gamma/p \rfloor$. We modify the SOLVED-BITS algorithm to return the least significant bit of the the value at no extra cost. Also since the least significant bits for both q and r associated with x_0 are random values, the chance of the algorithm not aborting is $1/4$. Therefore, if we need a constant round algorithm we need to run the following algorithm λ times in parallel to ensure that the chance of aborting in all runs negligible.

The algorithm for producing PK is given below (within, we refer to $\lfloor 2^\gamma/p \rfloor$ as $[x_\gamma]_B$):

Protocol 6 . $\langle x_0, \ldots, x_\tau \rangle \leftarrow \text{ComputeX}([x_\gamma]_B, [p], \tau, \rho)$,

```
1: For $i = 1..\tau$ run protocol $[q_i], [q_{i,0}] \leftarrow$ SOLVED-BITS$([x_\gamma]_B)$ in parallel.
2: For $i = 1..\tau$, and for $j = 0..\rho$ run protocol $[r_{i,j}] \leftarrow$ RAN$_2()$ in parallel.
3: For $i = 1..\tau$, locally compute $[r_i] \leftarrow (\sum_{j=0}^{\rho-1} [r_{i,j}] 2^j) \cdot (2[r_{i,\rho}] - [1])$.
4: For $i = 1..\tau$, run $[x_i] \leftarrow [p] \cdot [q_i] + r_i$ in parallel.
5: For $i = 1..\tau$, run protocol $x_i \leftarrow$ REVEAL$([x_i])$ in parallel.
6: $x_0 \leftarrow$ biggest of the revealed $x_i$'s.
7: Reveal the least significant bits from $q$ and $r$ in computing $x_0$. If either the former is not
   even, or the latter is not odd, abort.
8: Output $\langle x_0, \ldots, x_\tau \rangle$
```

Complexity Analysis Lines 1 and 2 can be run in parallel, and require 8 rounds and $\tau(52\ell + 24\sqrt{\ell} + 2\rho)$ multiplications. We need 2 multiplications in lines 3 and 4, but these can be done in parallel. Hence, the total round complexity would be 9, and the total number of multiplication invocations would be $\tau(52\ell + 24\sqrt{\ell} + 2\rho + 2)$. The largest number used in this protocol has at most γ bits, so the field order needs at least γ bits.

Simulation. For simplicity, in this paper we only show one execution of the simulator for the ComputeX protocol. Recall that if the least significant bits of q and r are not respectively even and odd, the algorithm should abort. Obviously these values for each run are public, and hence can be given as input to the simulator for as many times as it takes to get the final $\vec{x}$. The simulator $S_{\mathrm{X}}([e]^I, [p]^I, \vec{x'}, q_0, r_0)$ acts as follows:

1. The simulator calls $S_{\text{SOLVEDBITS}}([e]^I, 0)$ and learns $[q_i]^I$,
2. The simulator calls $S_{\text{RAN}_2}(0)$ for step 2 and learns $[r_i]^I$,
3. The simulator calls $S_{\text{MULT}}([p]^I, [q]^I, 0)$. Knowing $[r_i]^I$, the simulator learns $[x_i]^I$,
4. The simulator calls $S_{\text{REVEAL}}([x_i]^I, x'_i)$,
5. The simulator follows the step 6,

6. The simulator runs $S_{\textsc{reveal}}([q_0]^I, q_0)$ and $S_{\textsc{reveal}}([r_0]^I, r_0)$,

7. Output $\vec{x}'$.

4.4.6 Computing encryptions of $\vec{s}$

One step in Gentry's paradigm for FHE construction requires the public key to contain an encryption of the secret key. We assume circular security of the underlying encryption scheme, as do van Dijk et al. [vDGHV10] and Gentry[Gen09b]. Towards this goal, we design a protocol that enables players who hold private shares of the secret key (as well as the entire public key) to compute an encryption of the secret key under the public key. Note this cannot be done trivially with homomorphic evaluation because the encrypted secret-key is in fact necessary to homomorphically evaluate circuits of an arbitrary depth, resulting in a circular requirement. Similar issues arise when trying to produce a public-key for a leveled fully homomorphic encryption scheme.

Recall that in Dijk et al. [vDGHV10], the encryption of m under the public key $\langle x_0, \ldots, x_\tau \rangle$ computes as $[m + 2r + 2\sum_{i \in S} x_i]_{x_0}$, where $r \in (-2^{\rho'}, 2^{\rho'})$ and $S \subseteq \{1, \ldots, \tau\}$ is a random subset. Since both the x_i's and r can take negative values (as integers) whereas the computation is in a finite field, we need to somehow make sure the computation in the finite field result in the same integer value of the encryption of m. To resolve this issue, we compute the value *min* which is a unique value that satisfies the following two properties: 1) $min = 0 \mod x_0$, and 2) for an arbitrary S and for our set of x_i's and any value of r, it would make the summation $m + 2r + 2\sum_{i \in S} x_i$ positive. Because the range of values that r can take is public, all users can compute *min* locally and agree on respective shares. Next, to encrypt the secret key, all users generate shares for a set S and the shares for a value r. All users then add their shares of r, use shares in S to add in appropriate x_i's, and add *min* (see line 6 below). The players run the following protocol for each of the s_i's to obtain its encryption:

Protocol 7 . $(c_k) \leftarrow \text{EncryptS}([s_k], \langle x_0, \ldots, x_\tau \rangle)$

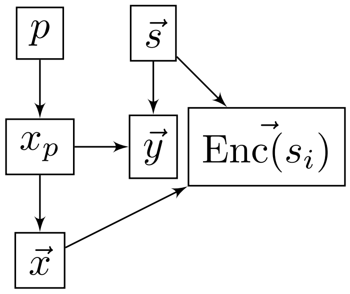
```
1: Locally compute $min$ as directed.
2: For $i = 0..\rho'$ run the protocol $[r_i] \leftarrow \textsc{ran}_2()$ in parallel.
3: For $i = 1..\tau$ run the protocol $[S_i] \leftarrow \textsc{ran}_2()$ in parallel.
4: Locally compute $[r] \leftarrow \sum_{i=0}^{\rho'-1} [r_i] 2^i$.
5: Run the protocol $[r] \leftarrow [r] \cdot (2 \cdot [r_{\rho'}] - [1])$.
6: Locally compute $[c'_k] \leftarrow [s_k] + [r] + 2\sum_i [S_i] \cdot x_i + [min]$.
7: Run the protocol $[c_k] \leftarrow \textsc{mod}([c'_k], x_0)$.
8: Run the protocol $c_k \leftarrow \textsc{reveal}([c_k])$
9: Output $c_k$.
```

Complexity Analysis The protocol produces bits in lines 2 and 3 which take 2 rounds of interaction (they can be run in parallel) and a total of $2(\rho' + \tau + 1)$ multiplication invocations. We perform another multiplication in line 5. The most expensive procedure, MOD, is called in line 7. Therefore,

the protocol needs 42 rounds of interaction and invokes the multiplication protocol $(31\ell^2 \log \ell + 31\ell \log \ell + 84\ell^2 + 71\ell + 36\ell\sqrt{\ell} + 30\sqrt{\ell} + 2(\rho' + \tau + 1) + 2)\Theta$ times.
Simulation. For $i \in [1, \Theta]]$, the simulator $\mathcal{S}_{\textbf{ENCS}}([s_i]^I, \vec{x}, c'_i)$ acts as follows:

1. For steps 2 and 3, the simulator calls $\mathcal{S}_{\textbf{RAN}_2}(0)$, and learns $[r_i]^I$ and $[S_i]^I$,
2. For step 5, the simulator calls $\mathcal{S}_{\textbf{MULT}}([r]^I, (2 \cdot [r_{\rho'}] - [1])^I, 0)$, and learns $[r]^I$,
3. Having shares of $[s_k]^I$, and $[r]^I$, the simulator locally computes $[c'_k]^I$,
4. For step 7, the simulator calls $\mathcal{S}_{\textbf{MOD}}([c'_k]^I, x_0, 0)$, and learn $[c_k]^I$,
5. For step 8, the simulator calls $\mathcal{S}_{\textbf{REVEAL}}([c_k]^I, c'_i)$,
6. Output c'_i.

Figure 1: Protocol hierarchy.



4.4.7 Complete generation protocol

We now put all of the pieces together and describe the entire key generation protocol. As we mentioned earlier, the public key consists of $\langle x_0, \ldots, x_\tau \rangle$, $\vec{y}$, and $\forall_i \overrightarrow{\textbf{Enc}}(s_i)$ which can be instantiated by calling the protocols ComputeX, ComputeY, and EncryptS. The secret key is the vector $\vec{s}$ which can be instantiated by calling the protocol ComputeS. To compute these values, we introduced two other helper protocols to generate the values p and x_p. For these two, we need to call protocols ComputeP and ComputeXP. Figure 1 shows the calling sequence for these protocols. The resulting protocol to generate (PK, SK) is as follows:

Protocol 8 . $(\text{PK}, \text{SK}) \leftarrow \text{SecureGen}(\eta, \tau, \rho, \theta, \Theta, \kappa)$

```
1: Call protocol $[p] \leftarrow \text{ComputeP}(\eta)$.
2: Call protocol $[\vec{s}], \theta' \leftarrow \text{ComputeS}(\theta, \Theta)$.
3: Call protocol $[x_p], [x_\gamma]_B \leftarrow \text{ComputeXP}([p], \kappa, \eta)$.
4: Call protocol $\vec{x} \leftarrow \text{ComputeX}([x_\gamma]_B, [p], \tau, \rho)$.
5: Call protocol $\vec{y} \leftarrow \text{ComputeY}([x_p], [\vec{s}], \theta')$.
```

Table 2: Complexity analysis of sub-protocols

	rounds	mult	field
p	2	$O(\eta)$	η
$\vec{s}$	7	$O(\Theta \log \Theta + \Theta\ell)$	$\log \Theta + 1$
x_p	23-49	$O(\ell \log \ell)$	$\eta\kappa + 2$
$\vec{x}$	9	$O(\ell\tau)$	γ
$\vec{y}$	30	$O(\ell \log \ell + \Theta\ell)$	$\lceil \log \theta(\kappa+1) \rceil + 1$
Enc(s_i)	42	$O(\Theta\ell^2 \log \ell)$	$2(\log \tau)\gamma$

```
6: For $i = 1..\Theta$, call protocol $c_i \leftarrow \text{EncryptS}([s_i], \vec{x}])$.
7: Output $\text{PK} = (\langle x_0, \ldots, x_\tau \rangle, \vec{y}, \vec{c})$, $\text{SK} = [\vec{s}]$.
```

Complexity Analysis Table 2 gives a summary of the round and multiplication complexity of each of the subprotocols. Note that the round complexity column for computing x_p is 23–49. This variance stems from the fact that the value of $[x_\gamma]_B$ is ready by the end of round 26 for the ComputeX protocol to start, and the x_p itself takes 57 rounds to complete for the ComputeY protocol to start. *If we run non-sequential protocols in parallel, the total number of rounds SecureGen needs is 102.*

Also the last column in Table 2 represents the minimum bit-length of ℓ (the field order) for each of the sub-protocols. The maximum of all these values is $\eta\kappa + 2$, which we can substitute as ℓ in all equations. Knowing ℓ, the multiplication invocation number would be the summation of the second column.

Simulation The simulator $S_{\mathbf{KEYGEN}}(\text{PK} = (\langle \vec{x}' \rangle, \vec{y}', \vec{c}'), q_0, r_0)$:

1. The simulator chooses a random p' which is an odd η-bit integer, and runs $S_{\mathbf{P}}(p')$ to learn $[p']^I$,
2. The simulator runs $S_{\mathbf{S}}(\vec{0})$ and learns $[\vec{s}']^I$,
3. The simulator runs $S_{\mathbf{XP}}([p']^I, 0)$ and learns $[xp]^I$ and $[x_\gamma]^I$,
4. The simulator runs $S_{\mathbf{X}}([x_\gamma]^I, [p']^I, \vec{x}', q_0, r_0)$,
5. The simulator runs $S_{\mathbf{Y}}([xp]^I, [\vec{s}']^I, \vec{y}')$,
6. For each $i \in [1, \Theta]$, the simulator calls $S_{\mathbf{ENCS}}(\vec{x}', [\vec{s}']^I, c_i')$,
7. Output $(\text{PK}, [\vec{s}']^I)$.

Theorem 4.2. *(Informal) The threshold scheme $\Pi = (\boldsymbol{Gen}, \boldsymbol{Enc}, \boldsymbol{Dec})$ described above satisfies the Threshold Indisitinguishability Security notion as per Def. 2.1 and the Circular Threshold Security notion as per Def. 2.2.*

Proof Sketch for Thm. 4.2 The indistinguishability and circular security for our scheme Π follows from the simulatability of the key generation procedure. We show how to transform an adversary A that has advantage ϵ in the Threshold indistinguishability game into an adversary A' for the FHE semantic security that also has advantage ϵ. The reduction is straightforward. Adversary $A'(1^k, pk)$, upon receiving a public key, runs the simulator for the keygen protocol with adversary A to begin an internal execution of the threshold security game. Adversary A eventually receives a set of C shares of a secret key (that are statistically independent of sk), and then produces a pair of messages. A' forwards these messages and then forwards the challenge ciphertext c^* to A, and finally echoes A's response as output. Notice that A' produces a statistical simulation of the threshold semantic security game owing to statistical security of the simulator for the threshold key generation procedure. Thus, the advantage of A' is also ϵ, and the security of the FHE scheme implies that ϵ must therefore be negligible in the security parameter k. Next, observe that the circular threshold security of the scheme can be argued in the same fashion.

4.5 Constant Round Decryption

Recall the encryption algorithm.

Enc$(m, \mathbf{PK})$ Generate a ciphertext c^* as before in **Enc**$'$ (see below). Then for $i \in 1, ..., \Theta$, set $z_i \leftarrow [c^* \cdot y_i]_2$, keeping only $\lceil \log \theta \rceil + 3$ bits of precision after the binary point for each z_i. Output both c^* and $\vec{z} = \langle z_1, ..., z_\Theta \rangle$.

Enc$'(pk, m \in \{0,1\})$ Choose a random subset $S \subseteq \{1, 2, ..., \tau\}$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$. Output $c^* \leftarrow [m + 2r + 2\sum_{i \in S} x_i]_{x_0}$.

We observe that for an arbitrary $\zeta = 2^{-z}$ for some integer z, there is a specific set of the FHE scheme's parameters that make $\sum_i s_i z_i$ within $\zeta < 1$ of an integer (in our setting, ζ is 1/4). Therefore, by basic properties of addition mod 2, we have:

$$\left[c^* - \left\lfloor \sum_i s_i z_i \right\rceil \right]_2 = \left[[c^*]_2 - \left[\left\lfloor \sum_i s_i z_i \right\rceil \right]_2 \right]_2 = \left[[c^*]_2 - \left[\left\lfloor \sum_i s_i z_i + \zeta \right\rfloor \right]_2 \right]_2$$

Since c^* is public, the decryption can be determined by revealing the first bit after binary point (i.e., the decimal point in a binary number) in $\sum_i s_i z_i + \zeta$. Assuming each party P_j has shares of $[s_i]$, the value $\sum_i s_i z_i + \zeta$ can be computed locally and revealed. But revealing bits other than the first bit after binary point in $\sum_i s_i z_i + \zeta$ might leak information about the secret key. Hence we need to distort all bits that are not relevant to the final answer.

In order to re-randomize all of the other bits that happen to be revealed, we add a random value r to $x = (\sum_i s_i z_i + \zeta)2^{k-1}$, where $k = \log \theta + 3$ (the multiplication by 2^{k-1} removes the binary point). We choose the field order p that we secret share in such that $0 \le x_{max} < p - 2^{|p|-1}$ (x_{max} is the maximum value x can take and is equal to $2 \cdot \theta \cdot 2^{\lceil \log \theta + 3 \rceil}$). The reason for choosing p in this way is to determine if a wrap-around happens in $x + r$ for some random number r based on the most significant bit of r and the result. More precisely, if the result of summation's most significant bit is 0 and if the most significant bit of r is one, then a modular reduction (i.e., wrap-around) occurred, and the result should be added to p. In step 7 we decide if such event occurred or not and we determine the integer value of the summation of $x + r$ in step 8.

Given the integer value of $x+r$ and knowing that the $(k-1)^{\text{th}}$bit of the result is 0, the k-th bit can be calculated as $\left[R''_k - r_k - r_{k-1}\overline{R''_{k-1}}\right]_2$. Steps 9 to 12 calculate this value securely.

Let $|z_i| = \log\theta + 4 = k$. The protocol for multiparty decryption of a ciphertext is as follows:

Protocol 9 . $m' \leftarrow \mathrm{Dec}(c^*, [s_1], \ldots, [s_n])$

```
1: Players locally compute $\vec{z}$ as directed. Let $\vec{z'} = 2^{k-1} \cdot \vec{z}$,
2: Run protocol$([r]_B, [r]_p) \leftarrow$SOLVED-BITS$()$,
3: Locally compute $[x]_p \leftarrow \sum_i [s_i] z'_i$,
4: Locally compute $[R]_p \leftarrow [x]_p + [r]_p$,
5: Run protocol $R \leftarrow$ REVEAL$([R]_p)$,
6: Locally compute $R' \leftarrow R + p$,
7: Locally compute $[c]_p \leftarrow [r_{\ell-1}]_p \overline{R_{\ell-1}}$,
8: For $i = 0, ..., k$ locally compute $[R''_i]_p \leftarrow [c]_p R'_i + ([1] - [c]_p) R_i$ in parallel,
9: Locally compute $[a]_p \leftarrow [R''_k]_p - [r_k]_p$,
10: Run protocol $[a']_p \leftarrow [a^2]_p$,
11: Locally compute $[a'']_p \leftarrow [a']_p - [r_{k-1}]_p \overline{R''_{k-1}}$,
12: Run protocol $[a''']_p \leftarrow [a''^2]_p$,
13: Run protocol $a''' \leftarrow$ REVEAL$[a''']_p$,
14: Output $m' \leftarrow [c^* - a''']_2$.
```

Notice that setting p as a Mersenne Prime decreases the round complexity. This is because the binary digits of such a prime are all 1. In step 2, the players collaborate on producing a random number in field p. The most expensive part of this step is to check if $r < p$ which takes 7 rounds. If p is Mersenne, we do not need such a check because the only case the produced r is not less than p is when r is equal to p.

Claim 4. ***Dec***$(sk, c^*) = [c^* - a''']_2$.

Proof. By the assumed setup, each player p_i holds the share $[s_i]$ for all $0 \le i \le \Theta$. Recall that

$$\mathbf{Dec}(sk, c^*) = \left[c^* - \left\lfloor \sum_i s_i z_i \right\rceil \right]_2$$

and that $k = \log\theta + 3$ and $x = \lfloor \sum_i s_i z_i \rceil 2^k$. All we need to prove is that $x_{k+1} = a'''$. Since c^* is public, revealing either x_{k+1} or m' would result in determining the other one.

By definition we have:

$$x \le \theta 2^{k+2},$$

and we have

$$0 \le x < p - 2^{|p|-1}$$

by the selection of parameter p. Instruction 2 defines a value $r \in [0, p]$ that is shared among the n players and instruction 4 defines $R \leftarrow x + r \bmod p$. Therefore, we have:

$$R'' = (x+r) = \begin{cases} R' = R + p & \textit{if } r > 2^{|p|-1} \text{ and } R < 2^{|p|-1} \\ R & o.w \end{cases}$$

When $r > 2^{|p|-1}$ and $R < 2^{|p|-1}$, it follows that $r_{|p|} = 1$ and $R_{|p|} = 0$ (i.e. the high-order bits of the values of r and R are 1 and 0 respectively). We conclude:

$$[R'']_B = [r_{|p|}] \cdot \overline{R_{|p|}} \cdot R' + \overline{\left[r_{|p|}\right]} \cdot R_{|p|} \cdot R$$

We have shown that R'' is the integer value of $x + r$. It is left to determine if a carry occurs between the bits k and $k+1$ in the addition $x + r$. If no carry happens, a''' is $\left[[R''_{k+1}] - [r_{k+1}]\right]_2$. If a carry between the mentioned bits happens, a''' is $\left[[R''_{k+1}] - [r_{k+1}] + 1\right]_2$. Let a_{b-0} denote the substring $a_b ... a_0$. To determine if a carry happens between the bits k and $k+1$, we consider the following cases:

1. Case 1: $[r]_{k,0} < 2^k$. As we mentioned earlier, $0 \leq [x]_{k,0} < 2^k$. Therefore if $x + r < 2^{k+1}$, then no carry occurs.
2. Case 2: $2^k \leq r_{k-0} < 2^{k+1}$. So:

   $$2^k < x_{k-0} + r_{k-0} < 2^{k+1} + 2^k$$

   The upper limit in the above equation means that both $k+1$th and kth bits of integer summation $x_{k-0} + r_{k-0}$ cannot be 1 at the same time. On the other hand, the lower limit guarantees that at least one of these bits is 1. Therefore, in this case a carry happens if and only if $R''_k = 0$.

Combining these results we conclude that $r_k\overline{R''_k}$ is 1 if a carry happens, and is 0 otherwise. Therefore:

$$a'''_0 = \left[[R''_{k+1}] - [r_{k+1}] - [r_k]\left[\overline{R''_k}\right]\right]_2$$

Instructions 9 to 12 computes the following value:

$$a''' = \left((R''_{k+1} - r_{k+1})^2 - r_k\overline{R''_k}\right)^2$$

Finally, we need to prove $a'''_0 = a'''$. A truth table verifies the equality. Also using truth tables it is easy to see that a''' reveals either 0 or 1 and no other value, (otherwise it might leak information regarding the variables r_{k+1}, $\overline{R''_k}$, R''_{k+1}, or r_k). □

4.5.1 Complexity Analysis

The protocol calls the SOLVED-BITS subprotocol in line 2 and calls multiplication in lines 10 and 12. Therefore, the protocol takes 9 rounds and requires $(52|p| + 24\sqrt{p} + 2)$ multiplications. From the key generation section above, recall that the field order needs to be $\eta\kappa + 2$. This constraint is still sufficient, as the only constraint needed here is that $0 \leq x_{max} < p - 2^{|p|-1}$.

4.6 Simulation

To prove security, we describe the required simulator S (ideal model adversary) that generates the view of real-life adversary. As usual, the simulator works by running an internal copy of the real-life adversary and an internal copy of the honest players. At a high level, the simulator extracts the shares of the real-life adversary, feeds this share to an ideal functionality which implements the k-th bit extractor function, and using the answer received, produces a view for the adversary which is consistent with the answer.

For adversary A, the simulator $S^A(1^k, c^*, [s_1]^I, \ldots, [s_i]^I, b^*)$ on input the security parameter, the ciphertext c^*, the shares of secret s known by adversary, and $b^* = \mathbf{Dec}(c^*, z)$ does the following:

1. For Step 2 of the protocol, the simulator S^A runs the sub-simulator $S_{\text{SOLVEDBITS}}()$ to produce the output r and its resolved bits. The simulator also learns $[r_i]^I$ and $[r]^I$.
2. For steps 3- 4 of the protocol, simulator S^A follows the protocol on behalf of the honest players that it simulates.
3. The simulator picks a random $\hat{x}$ which satisfies the following constraints: $\hat{x}$ should be a random number in the range of 0 and M, $\hat{x}/2^{k-1} - 1/4$ should be within $1/4^{\text{th}}$of an integer, and $\hat{x}_k = b^*$ (i.e., the k^{th}bit of $\hat{x}$ should be b^*).
4. Using $\vec{z}$, $[s_i]^I$, and $[r_i]^I$, simulator computes the shares that the adversary would hold as the shares of R, $[\vec{R_i}]^I$. The simulator then run the $S_{\text{REVEAL}}([\vec{R_i}]^I, (\hat{x} + r))$ sub-simulator to reveal $\hat{x} + r$ to the adversary.
5. Simulator follows the steps 6- 14 of the protocol.

4.7 Correctness of Simulation

The simulator and the real execution of the protocol only differ in steps 3 and 5 of *Dec* protocol. Since SOLVED-BITS and REVEAL protocols are secure protocols their simulations are indistinguishable from the real world executions. In step four, instead of outputting the real $x + r$, we reveal $\hat{x} + r$. Since r is a totally random number, therefore the output will be a random number as well and indistinguishable. Since $\hat{x}$ is taken from the same domain as x would be taken, therefore the result has the same correctness properties as $x + r$. The rest of the protocol is identical in both scenarios. As a result, the view of the adversary in a real execution is identical to view of the adversary in simulated execution.

Outputs of All Players. It is left to show that the output computed by all players in a real execution and in an ideal execution with an ideal adversary are identically distributed. The output of the simulation is identical to the output of the real execution, since *Dec* protocol reveals the $\log + 4^{\text{th}}$bit of x. As a result if this bit being set to be the same as the expected decrypted value, the output would be identical. Since the simulator has full control over setting the value of x the output will be identical to that of the real execution.

5 Secure Multiparty Computation

In this section, we follow the approach proposed by Cramer et al. [CDN01] for constructing a multi-party computation protocol based on threshold cryptography. Our biggest changes are that we do not need a protocol for multiplication, we use a different approach for proving knowledge of encryption, and we explicitly describe a key generation phase whereas it is assumed as an external setup in [CDN01]. As a consequence of requiring much less interaction among the parties, our simulation argument is somewhat simpler than the argument from [CDN01]. We use the same standard simulation-based definition of stand-alone secure multi-party computation. We assume the existence of a standard n-party CoinFlipping protocol which guarantees soundness in the presence of $< n/2$ adversaries: namely, for any minority set of adversaries, the protocol guarantees that the distribution is still statistically close to uniform. Such a protocol can be easily constructed based on the existence of hiding commitments. (Unlike CDN, we do not need this coin flipping protocol to be simulatable.)

5.1 Definition

In this section we adopt a standard the security definition for secure multi-party computation from [CDN01] and [IKK$^+$11]. This definitional approach compares the real-world execution of a protocol for computing a function with an ideal-world evaluation of the function by a trusted party. Security is then defined by requiring that for every adversary A attacking the real execution of the protocol there exists an ideal-world adversary A', sometimes referred to as a simulator, which "achieves the same effect" in the ideal world. This is made more precise in what follows.

The real model. Let π be a multi-party protocol computing a circuit f. We consider an execution of π on an open broadcast network with rushing in the presence of a statically-corrupting adversary A coordinated by a non-uniform environment $Z = \{Z_k\}$. At the outset of the execution, Z gives I and z to A, where $I \subset [n]$ represents the set of corrupted parties and z denotes an auxiliary input. Then the environment gives input x_i to each party P_i and gives $\{x_i\}_{i \in I}$ to A. The parties then run the protocol π with A providing the messages sent on behalf of any corrupted party. At the end of the execution, A gives to Z an output which is an arbitrary function of A's view thus far, and Z is additionally given the outputs of the honest parties. If the adversary aborts the protocol at some step (formally, if the output of some honest party at the end of the phase is $\perp$), execution is halted; otherwise, execution continues until the protocol is finished. Once the execution terminates, Z outputs a bit; we let $\mathrm{REAL}_{\pi,A,Z}(k)$ be a random variable denoting the value of this bit.

The ideal model. In the ideal model, there is a trusted party who computes f on behalf of the parties. This definition of ideal model corresponds to a notion of security where fairness and output delivery are guaranteed. Once again, we have an environment Z which provides inputs $x_1, \ldots, x_n$ to the parties, and provides $I, \{x_i\}_{i \in I}$, and z to A'. At the outset, Z gives I and z to A' and provides input x_i^j to party P_i and gives $\{x_i^j\}_{i \in I}$ to A'. Each honest party sends their input to the trusted party; adversary A' sends inputs on behalf of players in I and can also send the special symbol $\perp$ to the trusted party. The trusted party computes $y \leftarrow f(x_1, \ldots, x_n)$ using the inputs it receives from the players. For each player that submits $\perp$, the trusted party uses input 0. Finally, the trusted party delivers output y to each player who submits an input that is not $\perp$.

At the end of this phase, A' gives to Z an output which is an arbitrary function of its view thus far, and Z is additionally given the outputs of the honest parties. After all phases have been completed, Z outputs a bit. Once again, we let $\mathrm{IDEAL}_{\pi,A,Z}(k)$ be a random variable denoting the value of this bit. With the above in place, we can now define our notions of security.

Definition 5.1. *(Security) Let π be a multi-party protocol for computing a circuit f, and fix $s \in \{1, \ldots, n\}$. Then we say that π securely computes f in the presence of malicious adversaries corrupting s parties if for any ppt adversary A there exists a ppt adversary A' such that for every polynomial-size circuit family*

$Z = Z_k$ corrupting at most s parties the following is negligible:

$$\left|\Pr\left[\text{REAL}_{\pi,A,Z}(k) = 1\right] - \Pr\left[\text{IDEAL}_{f,A',Z}(k) = 1\right]\right|.$$

5.2 MPC Using Fully Homomorphic Encryption Scheme

In this section we present our protocol for evaluating an arbitrary circuit f in the presence of a minority of malicious adversaries. We assume that the players can communicate via an authenticated broadcast channel and via point-to-point private and authenticated channels (which may in turn be implemented using signatures, public key encryption, etc.)

Protocol 10 . Each party holds private input x_i; the parties jointly compute $f(x_1, \dots, x_n)$.

1: Party P_i receives as input $(1^k, n, x_i)$. (We assume the adversary receives as input $1^k, n$, a set of corrupted parties C and the inputs $\{x_c\}_{c \in X}$ for the corrupted parties, and auxiliary information.)

2: Players run the key generation subprotocol SecureGen$(\eta, \tau, \rho, \theta, \Theta, \kappa)$ to generate a public key pk and shares of the secret for the threshold scheme Π. At the end of this step, player p_i holds share $[s_i]$. If the sub-protocol halts prematurely, then players halt and output $\perp$.

3: The players take sequential turns sharing their input using the encryption scheme $\tilde{\Pi}$ that is constructed from Π (see §3.1.3). More specifically, for $i \in [1, n]$, player P_i broadcasts $c_{i,j} \leftarrow \tilde{\textbf{Enc}}(\tilde{\text{PK}}, x_{i,j})$. Then all of the players run a standard CoinFlipping protocol to generate a random string r_i. Player P_i now interprets r_i as n strings $r_{i,1}, \dots, r_{i,n}$ and uses coins $r_{i,j}$ as the random coins to run Verifier(PK,$c_{i,j}$) (see §3.1.3) of the Hidden Bit POK protocol on input $c_{i,j}$ for each bit $j \in [1, n]$ of input x_i. Player P_i runs the corresponding Prover algorithm on $c_{i,j}$ using the random coins used to generate $c_{i,j}$ as the witness, and broadcasts the Prover message. The remaining players also execute the Verifier algorithm using the same random coins and verify that the first message is consistent and the second message is accepted. If player P_i fails the POK protocol, then P_i is excluded from the rest of the protocol, and the remaining players that have not been excluded use a canonical encryption of 0 as the input for P_i (e.g., they use $\tilde{\textbf{Enc}}(\tilde{\text{PK}}, 0; 0)$ as each input bit).

4: The players that have not been excluded locally run $\textbf{Eval}(\text{PK}, c_{1,1}, \dots, c_{n,n}, \tilde{f})$ where the function $\tilde{f}$ first transforms the input ciphertexts encrypted under $\tilde{\Pi}$ into ones for scheme Π. This is done by homomorphically evaluating the decryption procedure described in §3.1.2.(Note: All of the ciphertexts in $c_{i,j}$ have a large degree of noise in them due to the circuit-privacy call that was used to rerandomize the ciphertexts. Therefore, the first thing that is done is that the ciphertexts are re-encoded with less noise using the same procedure as FHE bootstrapping.) Next, compute ciphertext z_i of the result $f(x_1, \dots, x_n)$. Note that each player can complete this step using only local information (since the public key for the FHE includes all the information needed for bootstrapping etc).

5: Each player P_i that has not been excluded broadcasts the ciphertext z_i computed in the previous step. Each player then locally computes the majority of the broadcasts as ciphertext z'. A majority is guaranteed to exist since the malicious players form a minority and **Eval** is deterministic. Any player whose broadcast differs from the majority is excluded from the remaining portion of the protocol.

```
6: Players $p_i$ that have not been excluded run the distributed subprotocol Dec$(z', [s_1], \ldots, [s_n])$
   using input $z'$ and their local share $[s_i]$. The output of the Dec protocol is taken as the
   output.
```

The Simulation We present a simulator A' for an adversary A that coordinates a group of corrupted players C. The steps of the simulation are as follows:

1. Begin an execution of the protocol for the adversary $A(1^k, n, C, \{x_c\}_{c \in C})$ in which the state for the honest players P_i is initialized with input $(1^k, n, x_i = 0)$ and is thereafter maintained internally by the simulator.

2. Run $(\text{SK}, \text{PK}) \leftarrow \mathbf{Gen}(1^k)$ to generate a public and secret key for the threshold FHE scheme. Run the simulator $S_{\mathbf{KEYGEN}}(\text{PK} = (\langle \vec{x'} \rangle, \vec{y'}, \vec{c'}))$ for the SecureGen key generation protocol with input PK using the adversary A and using the internal copies of the honest players. The states of the adversary A and the internal copies of the honest player are maintained at the end of the simulation.

3. For each honest player, use a randomly generated ciphertext under the public-key PK corresponding to 0 as its input, and follow the remaining procedure for running the coinflipping and hidden-bits POK protocol honestly. When it is time for a corrupted player $p_c \in C$ to run the POK, if the first execution succeeds, then invoke algorithm *Extractor* for the POK to recover the input x_c. If the first execution fails, then exclude the user from future simulated steps of the protocol. If the extractor fails, then abort the simulation.

4. Feed the inputs $\{x_c\}_{c \in C}$ for the corrupted players to the external trusted party and wait for a response output y.

5. Follow steps 4 and 5 of the Protocol on behalf of the honest players based on the broadcast input ciphertexts. After step 4, each internal copy of the honest player has computed a value z_i that they broadcast. In step 5, each internal honest copy broadcasts z_i; each corrupted player that does not broadcast z_i is excluded from the rest of the internal simulation. Compute z' for these honest players as per the protocol.

6. Run the simulator $S_{\mathbf{Dec}}(y, z')$ for the Dec protocol using A as an oracle for the malicious players to complete the internal simulation of the protocol for A.

7. Output whatever the adversary A outputs.

Theorem 5.2. *Let π be Protocol 8 for a function f, and fix $s \in \{1, \ldots, n/2\}$. Under the appropriate Approximate-GCD and sparse subset sum assumptions, it holds that for for any ppt adversary A, there exists a ppt adversary A' such that for every polynomial-size circuit family $Z = Z_k$ corrupting a minority of parties the following is negligible:*

$$\left| \Pr\left[\text{REAL}_{\pi,A,Z}(k) = 1\right] - \Pr\left[\text{IDEAL}_{f,A',Z}(k) = 1\right] \right|.$$

Proof. (High level Idea) We present a high-level summary of the changes needed in the security proof from [CDN01]. Our proof summary consists of a series of hybrid experiments that relate REAL and IDEAL and a brief description on why two consecutive hybrid experiments are indistinguishable.

$\mathbf{Hybrid}_1(1^k, A, Z)$**:** This hybrid experiment is the same as the real experiment REAL except that the experiment first generates $(\mathrm{SK}, \mathrm{PK}) \leftarrow \mathbf{Gen}(1^k)$ and then runs the simulator $S_{\mathbf{KEYGEN}}(\mathrm{PK} = (\langle \vec{x'} \rangle, \vec{y}', \vec{c'}))$ interacting with the adversary A.

We claim that REAL and $\mathbf{Hybrid}_1$ are identical because $S_{\mathbf{KEYGEN}}$ is information theoretically-secure.

$\mathbf{Hybrid}_2(1^k, A, Z)$**:** This hybrid experiment is the same as the previous one, except that the extractor for the Hidden Bit POK is used on each broadcast ciphertext from the adversary A. If any extraction fails, then the experiment aborts.

We claim that the $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_1$ distributions are statistically close. The only difference occurs when the extraction fails in the second hybrid for one instance of the Hidden Bit POK protocol. By the proof of knowledge extraction error property from the Hidden POK protocol proven in Thm. 3.11 and the union bound, it follows that these events occur with a negligibly small probability, and therefore the distributions are statistically close.

$\mathbf{Hybrid}_3(1^k, A, Z)$**:** This hybrid experiment is the same as the previous, except that the experiment sends the extracted input values for the malicious parties to the trusted party and receives output $y = f(x_1, \ldots, x_n)$ in return. The experiment then uses the simulator $S_{\mathbf{Dec}}(y, z)$ for the decryption on input (y, z) to force the players to output y. (Notice that at this point, z corresponds to an encryption of y, but that the simulator is used to feed messages to the adversary instead of the threshold decryption protocol.)

We claim that $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$ are computationally indistinguishable by the simulation property for the threshold decryption protocol and the unique decoding property of the POK protocol. In particular, the decoding property in Thm. 3.11 states that the inputs extracted from the adversaries will be the same as the inputs decrypted using SK (i.e., the inputs used in the real protocol computation). Thus, conditioned on this event that all inputs are consistent between the two experiments, the value y returned from the trusted party corresponds to the ciphertext z. Finally, the simulation property of $S_{\mathbf{Dec}}(y, z)$ guarantees that the transcripts between the two hybrids are identical.

$\mathbf{Hybrid}_4(1^k, A, Z)$**:** This hybrid experiment is the same as the previous, except that the input 0 is used to produce a ciphertext and run the Hidden Bit POK for each of the honest parties.

We claim that $\mathbf{Hybrid}_4$ and $\mathbf{Hybrid}_3$ are computationally indistinguishable based on the soundness of the hidden POK and the hidden bit property from Thm. 3.12. In particular, suppose that these two distributions were distinguishable with advantage ϵ. We define more hybrid experiments $\mathbf{Hybrid}_{3,i,j}$ in which all of the input bits up until the j^{th}bit of player i are formed using the 0 input, whereas the rest of the input bits use the honest player inputs. Note that $\mathbf{Hybrid}_{3,1,0} = \mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_{3,n,n} = \mathbf{Hybrid}_4$. Thus, there exists a pair of consecutive experiments $\mathbf{Hybrid}_{3,i,j}$ and $\mathbf{Hybrid}_{3,i,j+1}$ (without loss of generality, we assume this boundary occurs between j and $j+1$ instead of across the last bit of one player and the first bit of the next player and that the difference is between 0 and 1 in these positions) with advantage ϵ/n^2. For convenience, we denote this pair of hybrid experiments $\mathbf{Hybrid}_a$ and $\mathbf{Hybrid}_b$. We will now use this pair to violate the soundness of the POK, or the simulation properties.

We first claim that inputs $\{x_c\}^a$ extracted from the parties in C in $\mathbf{Hybrid}_a$ and the ones $\{x_c\}^b$ will be the same with all but negligible probability. (If extraction fails, we use $\perp$ to denote the extracted input. As argued earlier, the probability of extracting $\perp$ in $\mathbf{Hybrid}_3$ is negligible.) Suppose this is not true: i.e. $\Pr[\mathbf{Hybrid}_b(A) \text{ extracts } \{x_c\}^b \neq \{x_c\}^a] > \mu(k)$. Then there exists a vector of inputs $x = (x_1, \ldots, x_n)$ for which the probability that these two sets are different is greatest (inverse polynomial probability of success); let this vector, the position j, and the sets $\{x_c\}^a$ and $\{x_c\}^b$ be given as non-uniform advice for the following adversary A' that breaks the hidden POK property. The adversary A' runs the hybrid $\mathbf{Hybrid}_a$ using the inputs x while participating in the Hidden POK game. It receives a PK externally from the Hidden POK game and uses this PK with the simulator $S_{\mathbf{Gen}}$ in the first step of $\mathbf{Hybrid}_a$. It then receives a ciphertext under PK from the external game and uses it (along with its decryption opening query in the HB game) to run the (i,j) instance of the input bit protocol in $\mathbf{Hybrid}_a$. Finally, it runs the extractor for all of the malicious parties to recover a set of inputs $I = \{x_c\}_{c \in C}$. If $I = \{x_c\}^a$, then the adversary output 0, and otherwise outputs 1. Notice that if the input ciphertext is 0, then the adversary has run experiment $\mathbf{Hybrid}_a$, whereas if the input is 1, the adversary runs $\mathbf{Hybrid}_b$ and so:

$$\begin{aligned}\Pr[HB_{A'}(1^k) = 1] = &\Pr[b = 0] \Pr[A' \text{ outputs } 0 \mid b = 0] + \Pr[b = 1] \Pr[A' \text{ outputs } 1 \mid b = 1] \\ &\frac{1}{2} \Pr[\mathbf{Hybrid}_a(A) \text{ extracts } \{x_c\}^a] + \frac{1}{2} \Pr[\mathbf{Hybrid}_b(A) \text{ extracts } \{x_c\}^b \neq \{x_c\}^a] \\ &\geq \frac{1}{2} + \mu(k)/2\end{aligned}$$

Thus, our claim that the extracted outputs must be the same w.h.p holds. Conditioned on this event that both extracted sets are equal, it follows that the value y recovered in $\mathbf{Hybrid}_4$ and the decryption of z from $\mathbf{Hybrid}_3$ will be the same. However, in $\mathbf{Hybrid}_4$, the ciphertext z corresponds to a different plaintext, namely $f(\{x_c\}^a, 0, \ldots, 0)$ where 0 is used for the honest players. We now claim that this difference is indistinguishable by introducing another hybrid experiment $\mathbf{Hybrid}_{3z}$ in which the same ciphertext from $\mathbf{Hybrid}_3$ is given to the experiment and used in place of the encryption generated in $\mathbf{Hybrid}_4$. We claim—through a standard argument—that $\mathbf{Hybrid}_{3z}$ and $\mathbf{Hybrid}_4$ must be indistinguishable based on the semantic security of the threshold encryption scheme. The only remaining difference to account for is the use of the simulator $S_{\mathbf{Dec}}$ for decryption. The stand-alone security of this simulator implies our claim.

□

References

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *EUROCRYPT*, pages 1–35, 2009. 5, 7

[BIB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *PODC*, pages 201–209, 1989. 14

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10. ACM, 1988. 12

[CDD$^+$99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer-Verlag, May 1999. 7

[CDN01] Cramer, Damgard, and Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 2001. 1, 2, 3, 4, 30, 31, 33

[Des87] Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO: Proceedings of Crypto*, 1987. 2

[DF89] Desmedt and Frankel. Threshold cryptosystems. In *CRYPTO: Proceedings of Crypto*, 1989. 2

[DFK$^+$06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC*, pages 285–304, 2006. 12, 13, 14

[DJ01] Damgard and Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC: International Workshop on Practice and Theory in Public Key Cryptography*. LNCS, 2001. 2

[ElG85] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31:469–472, 1985. 2

[FGMY97a] Frankel, Gemmell, MacKenzie, and Yung. Optimal-resilience proactive public-key cryptosystems. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1997. 2

[FGMY97b] Frankel, Gemmell, MacKenzie, and Yung. Proactive RSA. In *CRYPTO: Proceedings of Crypto*, 1997. 2

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig. 1, 2, 6

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009. 24

[GH110] Cpa and cca-secure encryption systems that are not 2-circular secure, 2010. matthewdgreen@gmail.com 14686 received 16 Mar 2010, last revised 18 Mar 2010. 3

[HLOV09] Brett Hemenway, Benoit Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. Technical Report 2009/088, eprint.iacr.org, 2009. Cryptology ePrint Archive. 4, 5, 7

[IKK$^+$11] Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM J. Comput*, 40(1):122–141, 2011. 31

[JJ00] Jakobsson and Juels. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 2000. 2

[KLML05] Kiltz, Leander, and Malone-Lee. Secure computation of the mean and related statistics. In *Theory of Cryptography Conference (TCC), LNCS*, volume 2, 2005. 17

[NN01a] Moni Naor and Kobbi Nissim. Communication complexity and secure function evaluation. *CoRR*, cs.CR/0109011, 2001. 1

[NN01b] Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 590–599, New York, NY, USA, 2001. ACM. 2

[NO07] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2007. 13

[Rab98] T. Rabin. A simplified approach to threshold and proactive RSA. *Lecture Notes in Computer Science*, 1462:89–??, 1998. 2

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), 1979. 2, 12

[Tof09] Tomas Toft. Constant-rounds, almost-linear bit-decomposition of secret shared values. In Marc Fischlin, editor, *Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*, volume 5473 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2009. 13, 14

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010. 1, 2, 6, 14, 15, 16, 20, 24