Towards Robust Computation on Encrypted Data

Manoj Prabhakaran* and Mike Rosulek*

University of Illinois, Urbana-Champaign
{mmp,rosulek}@uiuc.edu

Abstract. Encryption schemes that support computation on encrypted data are useful in constructing efficient and intuitively simple cryptographic protocols. However, the approach was previously limited to stand-alone and/or honest-but-curious security. In this work, we apply recent results on "non-malleable homomorphic encryption" to construct new protocols with Universally Composable security against active corruption, for certain interesting tasks. Also, we use our techniques to develop non-malleable homomorphic encryption that can handle homomorphic operations involving more than one ciphertext.

1 Introduction

Computation on encrypted data is one of the most intriguing problems in cryptography today. There is a long history of works investigating this problem in various general settings [22, 12, 2, 1, 11, 23, 3, 13, 5, 17], as well as in relation to specific computational tasks (e.g., searching on encrypted inputs [10, 24, 14, 4, 18, 15, 19, 13, 8]). As demonstrated by these works, being able to compute on encrypted inputs leads to simple intuitive protocols for many cryptographic tasks.

However, compared to some of the core areas in cryptography like encryption, authentication and secure multi-party computation, the state of the art for computation on encrypted inputs remains quite limited. The majority of encryption schemes that allow computations on encrypted data are only known to achieve security against chosen-plaintext attacks. As such, protocols that manipulate encrypted data often have to employ complicated machinery of zero-knowledge proofs and/or distributed key management to provide protection against malicious participants. Similarly, issues like composability of protocols have hardly been explored for this problem.

In this work we take a closer look at the composability and non-malleability aspects of computation on encrypted data. Our goal is to construct protocols that are secure in the demanding setting of Universally Composable (UC) security [7]. The main challenge is in forbidding a malicious party from manipulating encrypted data in unwanted ways. The traditional solution to this problem is to use zero-knowledge proofs to enforce honest behavior. However, general zero-knowledge proofs are not possible in the UC framework.

* Partially supported by NSF grant CNS 07-47027.

Instead, our approach is to restrict malicious parties' capabilities via strong non-malleable guarantees on the encryption scheme itself. This approach has the additional benefit that shifting some of the security burden to the encryption scheme allows us to construct conceptually simple protocols that still achieve strong security against malicious parties.

Requiring "non-malleability" for an encryption scheme may seem counterproductive to the goal of computing on its encrypted data. Indeed, a scheme must necessarily be malleable in some way for its encrypted data to be manipulated. However, a security notion called *Homomorphic-CCA (HCCA)* security has recently been defined in [20], meaningfully combining homomorphic computational features and non-malleability. Briefly, a scheme that achieves HCCA security is homomorphic with respect to certain operations, but explicitly forbids all other manipulations to the underlying plaintext.

The HCCA security requirement is strong enough to be meaningful in the UC framework, but unlike general-purpose UC zero-knowledge proofs, can be achieved in the plain model. Indeed, such a scheme has been constructed in [20], under a standard assumption. However, that construction only supports a very limited class of homomorphic operations. In particular, it does not support operations which combine multiple encrypted inputs, which are relevant in the context of computation on encrypted data. Our contribution in this work is to show that when used with appropriately encoded data, the relatively unexpressive scheme from [20] can be used to robustly implement more sophisticated computations on data encrypted in multiple ciphertexts.

1.1 Overview of Our Results

Background: Non-Malleable Homomorphic Encryptions. Computation on encrypted data necessitates having an encryption scheme that supports some homomorphic operations. However, when considering security against malicious parties, a non-malleability requirement is also generally needed.

A key component in our constructions is a public-key encryption scheme that meaningfully combines both non-malleability and homomorphic operations. Such schemes were introduced in [20]. We review the relevant security definitions for these schemes in Section 2. For the purposes of this overview, the reader may consider a "non-malleable (unary) homomorphic encryption scheme" to be one in which the *only* ways to construct a valid ciphertext are: (1) encrypting a known message, or (2) applying a homomorphic operation to some $\mathsf{Enc}(m)$ to obtain $\mathsf{Enc}(T(m))$, for any function T in a set of *allowed transformations*. The set of allowed transformations is a fixed parameter of the encryption scheme, and it is infeasible for an adversary to generate a ciphertext whose value depends on other ciphertexts in any other way. Furthermore, ciphertexts derived via the homomorphic operation are completely indistinguishable (even to the recipient) from ciphertexts generated by the standard encryption operation. In [20], a construction was given for a family of encryption schemes that support these requirements for a range of allowed transformation operations related to

cyclic group operations. Our results do not rely on any additional properties of that construction, but uses the primitive in a black-box manner, and as such, can be instantiated with the construction in [20] or any future construction satisfying the appropriate security requirements.

The common technique in our constructions is to exploit the power of this encryption scheme as follows: We encode the input data with some special randomized "integrity" information into a vector of several ciphertexts. The integrity information is intended to correlate the vector of ciphertexts together into one "bundle." The homomorphic property of the scheme ensures that the integrity information and data can be manipulated in certain ways. For instance, in both of our main results, the integrity information can be "re-randomized" using the scheme's homomorphic operations.

When using a homomorphic non-malleable encryption scheme in a protocol, already by the non-malleability property of the encryption scheme, ciphertexts can only be derived from others using a certain limited class of operations. By employing an appropriate integrity encoding, we further enforce that among the small set of allowed operations, the only ones which preserve/maintain the integrity information are the legitimate operations prescribed by the protocol. In other words, the integrity encoding provides a means to give and verify an implicit zero-knowledge proof that the protocol is being honestly implemented.

Opinion Polling. Our first result is an "opinion poll" protocol that elegantly illustrates the power of the combination of non-malleability, unlinkability and homomorphism in a single encryption scheme. The protocol is motivated by the following scenario: A pollster wishes to collect information from many respondents. However, the respondents are concerned about the anonymity of their responses. Indeed, it is in the interest of the pollster to set things up so that the respondents are guaranteed anonymity, especially if the subject of the poll is sensitive personal information.

To help collect responses anonymously, the pollster can enlist the help of an external tabulator. The respondents require that the external tabulator too does not see their responses, and that if the tabulator is honest, then responses are anonymized for the pollster (i.e., so that he cannot link responses to respondents). The pollster, on the other hand, does not want to trust the tabulator at all: if the tabulator tries to modify any responses, the pollster should be able to detect this so that the poll can be invalidated.

A relevant view of this problem is as an instance of a model that we call *crypto-computing on third-party inputs* — a model that extends the "crypto-computing" model from [23]. In this new model, the inputs to the computation are owned by a set of parties other than the client (who receives the output — the pollster in our case) and the server (who does the actual computation on encrypted data — the tabulator in our case). This separation of roles introduces new security requirements: (1) Privacy for the input parties: the client should not learn anything other than the intended output value. The server should not learn anything either. (The input providers are not necessarily interested in the

correctness of the computation.) (2) Robustness: a malicious server cannot make the client accept an output that is inconsistent with the parties' inputs.

The opinion poll scenario is similar to the classic setting for mix-nets [9], where a group of servers accepts a list of ciphertexts and outputs a random permutation of their decrypted values. However, in many mix-net protocols it can be quite complicated to enforce the correctness of outputs against a malicious (i.e., actively corrupt) server (in our case, the tabulator in particular). Often zero-knowledge proofs [16], or distributed decryption via verifiable secret sharing are used to enforce the integrity of operations performed on the ciphertexts. In contrast, our use of non-malleable homomorphic encryption leads to a simple and elegant UC-secure protocol.

The main idea in our protocol is to use an encryption scheme whose *only* homomorphic operation is $\mathsf{Enc}(\alpha, \beta) \mapsto \mathsf{Enc}(\alpha, t\beta)$, where t, α, β are elements of some cyclic group. In other words, plaintexts consist of a pair of group elements. Anyone can multiply (apply the group operation to) the second plaintext component with a known value t, but the first component is completely non-malleable, and the two components remain "tied together." Now, to implement the opinion poll protocol, the pollster generates a (multiplicative) secret sharing $r_1, \ldots, r_n$ of a random secret group element R, then sends to the ith respondent a share r_i. Each respondent sends $\mathsf{Enc}(m_i, r_i)$ to the tabulator, where m_i is his response to the poll. Now the tabulator can blindly re-randomize the shares (multiply the ith share by a random s_i, such that $\prod_i s_i = 1$), shuffle the resulting ciphertexts, and send them to the pollster. The pollster will ensure that the shares encode the secret R and accept the results.

Informally, security is argued as follows. The pollster only sees a random permutation of the responses, and since the multiplicative sharing of R is re-randomized, there is no way to link any responses to the r_i shares he originally dealt to the respondents. The tabulator sees only encrypted data, and in particular has no information about the secret R or any individual shares r_i. The only way the tabulator could successfully (with non-negligible probability) generate ciphertexts whose second components are a multiplicative share of R is by making exactly one of his ciphertexts be derived from each respondent's ciphertext. By the non-malleability of the encryption scheme, each response m_i is inextricably "tied to" the corresponding share r_i and cannot be modified, so each respondent's response should be represented exactly once in the tabulator's output. Finally, observe that the responses of malicious respondents must be independent of honest parties' responses – by "copying" an honest respondent's ciphertext to the tabulator, a malicious respondent also "copies" the corresponding r_i. The resulting shares would be inconsistent with overwhelming probability.

We also show a similar protocol where the computation performed is a boolean-OR of the respondents' boolean inputs (where the tabulator also provides an input). Again, the non-triviality in these constructions is not in the complexity of the computation performed, but in ensuring (using only the properties of the encryption scheme, and in particular no zero-knowledge proofs) that a malicious server cannot do anything unwanted without detection.

Binary Homomorphic Encryption. Our second contribution is an extension of the non-malleable homomorphic encryption scheme of [20]. The scheme of [20] is homomorphic in an inherently *unary* way; it prohibits operations that combine multiple ciphertexts together in a homomorphic way. However, many existing applications of (plain) homomorphic encryption schemes rely on combining multiple ciphertexts together. Unfortunately, in [20], it was shown that it is impossible to achieve the natural extension of the security definitions to the setting where the homomorphic operations act on multiple ciphertexts. The complication arose from the tension between the non-malleability requirement and the *unlinkability* requirement (namely, that a ciphertext not leak whether it was derived as a normal encryption or via one of the homomorphic operations).

In this work, we show that a meaningful relaxation of these definitions can be achieved. Instead of settling for absolute unlinkability, we consider a relaxation similar to that used in [23], in which ciphertexts grow in size after applying the operations. Thus, a ciphertext will reveal no more than (an upper bound on) the number of homomorphic operations that have been applied to derive it. However, unlike in [23], our goal is to achieve non-malleability and robustness against malicious adversaries.

We construct an encryption scheme that supports the binary group operation in a cyclic group; i.e., anyone can transform $\mathsf{Enc}^*(\alpha)$ and $\mathsf{Enc}^*(\beta)$ into $\mathsf{Enc}^*(\alpha\beta)$, but the scheme is otherwise non-malleable. Lacking a "standard" security definition for such an encryption scheme, we prove that our construction is a UC-secure realization of a natural ideal functionality, whose details are motivated by extending the UC functionality considered in [20].

The main idea in our construction is to encode a message m as a vector $\mathsf{Enc}(m_1), \ldots, \mathsf{Enc}(m_k)$, where the m_i's are a random multiplicative sharing of m in the group. and Enc is a non-malleable homomorphic encryption scheme that supports (unary) group operations (from [20]). To "multiply" two such encrypted encodings, we can simply concatenate the two vectors of ciphertexts together, and rerandomize the new set of shares (multiply each component by s_i, where $\prod_i s_i = 1$, as in the opinion poll protocol) to bind the sets together.

The above approach captures the main intuition, but our actual construction uses a slightly different approach to ensure UC security. In the scheme described above, anyone can split the vector $\mathsf{Enc}(m_1), \ldots, \mathsf{Enc}(m_k)$ into two smaller vectors that encode two (random) elements whose product is m. We interpret this as a violation of our desired properties, since it is a way to make two encodings whose values are related to a *longer* encoding. To get around this problem of "breaking apart" these ciphertexts, we encode m as $\mathsf{Enc}(\alpha_1, \beta_1), \ldots, \mathsf{Enc}(\alpha_k, \beta_k)$, where the α_i's and β_i's form two *independently random* secret sharings of m. Rerandomizing these encodings is possible when we use a scheme that is homomorphic with respect to the operations $(\alpha, \beta) \mapsto (t\alpha, s\beta)$. Now these encodings cannot be split up in such a way that the first components and second components are shares of the same value. Note that it is crucial here that because of the non-malleability properties of the scheme, the (α_i, β_i) pairs cannot themselves be "broken apart."

2 Preliminaries

Homomorphic Encryption Syntax and Security. Our constructions use homomorphic encryption schemes that have *unary* homomorphic operations on the plaintext messages. That is, we suppose there is a procedure CTrans, which takes a ciphertext and a (description) of a function T on plaintexts, such that $\mathsf{Dec}_{SK}(\mathsf{CTrans}(\zeta, T)) = T(\mathsf{Dec}_{SK}(\zeta))$ is satisfied.

Prabhakaran and Rosulek [20] introduced security definitions for homomorphic encryptions that combine non-malleability as well as robust homomorphic features. Schemes satisfying these definitions are vital for achieving UC security in our constructions. We present a high-level overview of their security definitions below; we refer the reader to Appendix A for the complete formal definitions.

Informally, a homomorphic encryption scheme achieves *Homomorphic-CCA (HCCA) security* with respect to a set of functions $\mathcal{T}$ if the scheme is non-malleable except for the possibility of changing an encryption of m into an encryption of $T(m)$, for $T \in \mathcal{T}$ (i.e., no other operations are possible in the scheme). We also consider the complementary requirement: Informally, a homomorphic scheme is *unlinkable* with respect to $\mathcal{T}$ if it is indeed possible to change encryptions of m into encryptions of $T(m)$ for $T \in \mathcal{T}$ as a feature (using the CTrans operation), in such a way that ciphertexts do not reveal whether they were generated via Enc or via CTrans.

Formalizing the intuitive HCCA requirement in a general way is non-trivial. It is achieved in [20] by requiring that there be an additional procedure RigEnc_{PK} (used only in the analysis) which outputs a special "rigged" ciphertext ζ and some auxiliary information S, such that ζ is indistinguishable from a normal ciphertext. The rigged ciphertext does not necessarily encode a message; however, there is a corresponding procedure $\mathsf{RigExtract}_{SK}$ which, when given another ciphertext ζ' and the auxiliary information S, determines whether ζ' was obtained by applying a transformation to ζ, and if so, outputs that transformation. The formal HCCA security experiment enforces the indistinguishability of rigged and normal ciphertexts, as well as the correctness of $\mathsf{RigExtract}$'s output. Intuitively, if $\mathsf{RigExtract}$ only outputs transformations in $\mathcal{T}$, then ciphertexts can only depend on the values of other ciphertexts according to transformations in $\mathcal{T}$.

The unlinkability requirement is formalized via a more straight-forward security experiment. At a high level, the experiment enforces that for all adversarially generated ciphertexts ζ such that $\mathsf{Dec}_{SK}(\zeta) \neq \perp$, the two distributions $\mathsf{Enc}_{PK}(T(\mathsf{Dec}_{SK}(\zeta)))$ and $\mathsf{CTrans}(\zeta, T)$ are indistinguishable, even in the presence of a decryption oracle.

Concrete constructions. Prabhakaran and Rosulek [20] give a construction achieving the desired properties for various kinds of homomorphic operations, under the Decisional Diffie-Hellman assumption.

Let $\mathbb{G}$ be a cyclic group, and let $\mathbb{G}^n$ denote the product group, where we extend the group operation in $\mathbb{G}$ component-wise. For $\sigma \in \mathbb{G}^n$, define the function $T_\sigma : \mathbb{G}^n \to \mathbb{G}^n$ as the "multiplication by σ" operation: $T_\sigma(\alpha) = \sigma\alpha$. Finally, for any $\mathbb{H} \subseteq \mathbb{G}^n$, define $\mathcal{T}_{\mathbb{H}} = \{T_\sigma \mid \sigma \in \mathbb{H}\}$.

Theorem 1 ([20]). *For any $n \geq 1$ and any subgroup $\mathbb{H}$ of $\mathbb{G}^n$, there is an encryption scheme with message space $\mathbb{G}^n$ that is simultaneously HCCA-secure and unlinkable, with $\mathcal{T}_{\mathbb{H}}$ as the set of allowed operations, provided that the Decisional Diffie-Hellman (DDH) assumption holds in $\mathbb{G}$ and any subgroup of $\mathbb{Z}^*_{|\mathbb{G}|}$.*

Our two main results use instantiations of the above construction with $n = 2$, and $\mathbb{H} = \{1\} \times \mathbb{G}$ and $\mathbb{H} = \mathbb{G}^2$, respectively.

3 Opinion Polling

We describe an intuitively simple yet robust protocol for the opinion polling application described in Section 1.1, using HCCA encryption as a component.

Formally, we give a secure protocol for the UC ideal functionality $\mathcal{F}_{\mathsf{poll}}$, described in Figure 1. For the opinion polling application, we associate the pollster with party P_{client}, the tabulator with P_{server}, and the respondents with the input parties $P_1, \ldots, P_n$. Note that in $\mathcal{F}_{\mathsf{poll}}$, P_{client} learns only a random permutation of the parties' inputs, while P_{server} learns nothing about their inputs (except the knowledge of who has submitted inputs). Also, P_{server} and each input party can cause the process to abort without P_{client} accepting any output.

On input $[\text{SETUP}, P_{\mathsf{client}}, P_{\mathsf{server}}, P_1, \ldots, P_n]$ from party P_{client}:
- Send $[\text{SETUP}, P_{\mathsf{client}}, P_{\mathsf{server}}]$ to each party P_i.
- Send $[\text{SETUP}, P_{\mathsf{client}}, P_1, \ldots, P_n]$ to P_{server}.

On input $[\text{INPUT}, x_i]$ from input party P_i:
- Send $[\text{INPUTFROM}, P_i]$ to P_{server}, and remember x_i.

On input "OK" from P_{server}:
- If P_{server} is corrupt, expect to receive from P_{server} a permutation σ on $\{1, \ldots, n\}$. If P_{server} is honest, choose σ at random.
- If not all $P_1, \ldots, P_n$ parties have supplied an input, or if some $x_i = \perp$, then send $\perp$ to P_{client}.
- Otherwise, give $(x_{\sigma(1)}, \ldots, x_{\sigma(n)})$ to P_{client}.

On input "CANCEL" from a corrupt P_{server}, send $\perp$ to P_{client}.

Fig. 1. UC ideal functionality $\mathcal{F}_{\mathsf{poll}}$.

The Protocol. We present our protocol for $\mathcal{F}_{\mathsf{poll}}$ following the high-level overview given in Section 1.1. We then prove that the protocol is a UC-secure realization of $\mathcal{F}_{\mathsf{poll}}$, provided that at least one of $\{P_{\mathsf{client}}, P_{\mathsf{server}}\}$ are honest.

Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CTrans})$ be an unlinkable HCCA-secure scheme, whose message space is $\mathbb{G}^2$ for a cyclic group $\mathbb{G}$, and whose allowed (unary) transformations are $(\alpha, \beta) \mapsto (\alpha, t\beta)$ for all $t \in \mathbb{G}$. We suppose the CTrans operation accepts arguments as $\mathsf{CTrans}(C, t)$, where $t \in \mathbb{G}$ specifies the transformation $(\alpha, \beta) \mapsto (\alpha, t\beta)$. We abbreviate the $\mathsf{CTrans}(C, t)$ operation as "$t * C$". Thus $t * \mathsf{Enc}_{PK}(\alpha, \beta)$ is indistinguishable from $\mathsf{Enc}_{PK}(\alpha, t\beta)$, in the sense of the unlinkability definition.

The protocol proceeds as follows:

1. P_{client} generates a key pair $(SK, PK) \leftarrow \mathsf{KeyGen}$ and chooses random elements $r_1, \ldots, r_n \leftarrow \mathbb{G}$, remembering $R = \prod_i r_i$. She then sends $(PK, r_i, P_{\mathsf{server}})$ to each party P_i, and sends $(P_{\mathsf{client}}, P_1, \ldots, P_n)$ to P_{server}.
2. Input party P_i holds input x_i. He receives $(PK, r_i, P_{\mathsf{server}})$ from P_{client}, then sends $\mathsf{Enc}_{PK}(x_i, r_i)$ to P_{server} through a secure channel.
3. P_{server} collects ciphertext C_i from each input party P_i, then chooses a random permutation σ on $[n]$ and random $s_1, \ldots, s_n \leftarrow \mathbb{G}$ subject to $\prod_i s_i = 1$. He computes $C'_i = s_{\sigma(i)} * C_{\sigma(i)}$ and sends $(C'_1, \ldots, C'_n)$ to P_{client}.
4. P_{client} decrypts each C'_i as $(x'_i, r'_i) \leftarrow \mathsf{Dec}_{SK}(C'_i)$. If any decryptions fail, or if $\prod_i r'_i \neq R$, she aborts. Otherwise, she outputs $(x'_1, \ldots, x'_n) = (x_{\sigma(1)}, \ldots, x_{\sigma(n)})$.

Theorem 2. *If $\mathcal{E}$ is unlinkable and HCCA-secure with message space $\mathbb{G}^2$, and allowed transformations as described above, where $|\mathbb{G}|$ is superpolynomial in the security parameter, then our protocol is a secure realization (with respect to static corruptions) of $\mathcal{F}_{\mathsf{poll}}$, against adversaries who corrupt at most one of $\{P_{\mathsf{server}}, P_{\mathsf{client}}\}$.*

Proof. Given a real-world adversary $\mathcal{A}$, we construct a simulator $\mathcal{S}$. We break the proof down into 3 cases according to which parties $\mathcal{A}$ corrupts:

Case 1: If $\mathcal{A}$ corrupts neither P_{server} nor P_{client}, then suppose by symmetry that $\mathcal{A}$ corrupts some input parties $P_1, \ldots, P_k$. Then the main task for $\mathcal{S}$ is to extract the inputs of each corrupt P_i and send them to $\mathcal{F}_{\mathsf{poll}}$. $\mathcal{S}$ simply does the following:

- On receiving $[\textsc{setup}, P_{\mathsf{client}}, P_{\mathsf{server}}, P_1, \ldots, P_n]$ from $\mathcal{F}_{\mathsf{poll}}$, generate $(PK, SK) \leftarrow \mathsf{KeyGen}$. Choose random $r_1, \ldots, r_k \leftarrow \mathbb{G}$ and simulate that P_{client} sent $(PK, r_i, P_{\mathsf{server}})$ to each corrupt input party P_i.
- If not all corrupt parties P_i send a ciphertext C_i to P_{server}, then abort. Otherwise, set $(x_i, r'_i) \leftarrow \mathsf{Dec}_{SK}(C_i)$.
- If any of the above decryption fails, or if $\prod_i r'_i \neq \prod_i r_i$, then send $[\textsc{input}, \perp]$ to $\mathcal{F}_{\mathsf{poll}}$ on behalf of *each* corrupt input party P_i.
- Otherwise send $[\textsc{input}, x_i]$ to $\mathcal{F}_{\mathsf{poll}}$ on behalf of each corrupt input party P_i.

It is straight-forward to see that in the cases where $\mathcal{S}$ sends $[\textsc{input}, \perp]$, then by the honest behavior of P_{server} and P_{client}, the protocol would have mandated that P_{client} refuse the output.

Case 2: If $\mathcal{A}$ corrupts P_{client} and (without loss of generality) input parties $P_1, \ldots, P_k$, then $\mathcal{S}$ does the following:

- When corrupt P_{client} sends $(PK, r_i, P_{\mathsf{server}})$ to each honest input party P_i, send $[\textsc{setup}, P_{\mathsf{client}}, P_{\mathsf{server}}, P_1, \ldots, P_n]$ to $\mathcal{F}_{\mathsf{poll}}$ on behalf of P_{client}.
- When a corrupt input party P_i sends a ciphertext C_i to honest P_{server}, send $[\textsc{input}, 1]$ to $\mathcal{F}_{\mathsf{poll}}$ on behalf of P_i.
- When $\mathcal{F}_{\mathsf{poll}}$ gives the final output to $\mathcal{S}$, remove as many 1's from the output list as there are corrupt input parties. Call the remaining outputs $x_{k+1}, \ldots, x_n$. Honestly simulate the remainder of the protocol on behalf of the honest input parties, using x_i as the input for honest party P_i.

Since P_{client} is corrupt, $\mathcal{S}$ can legally obtain the set of honest input parties' inputs. The only difference therefore between the view of $\mathcal{A}$ in the real world and our simulation is that the honest parties are simulated with inputs that may be permuted. However, since P_{server} is honest, P_{client}'s view in the protocol is independent of any permutation on the honest parties' inputs.

Case 3: If $\mathcal{A}$ corrupts P_{server} and input parties $P_1, \ldots, P_k$, then $\mathcal{S}$ does the following:

- When $\mathcal{F}_{\mathsf{poll}}$ gives $[\text{SETUP}, P_{\mathsf{client}}, P_1, \ldots, P_n]$ to $\mathcal{S}$, generate $(PK, SK) \leftarrow \mathsf{KeyGen}$. Pick random $r_1, \ldots, r_n \leftarrow \mathbb{G}$ and simulate that P_{client} sent $(PK, r_i, P_{\mathsf{server}})$ to each corrupt P_i.
- When $\mathcal{F}_{\mathsf{poll}}$ gives $[\text{INPUTFROM}, P_i]$ to $\mathcal{S}$ for an honest party ($i > k$), generate $(C_i, S_i) \leftarrow \mathsf{RigEnc}_{PK}$ and simulate that P_i sent C_i to P_{server}. Remember S_i.
- When P_{server} sends P_{client} a list of ciphertexts $(C'_1, \ldots, C'_n)$, do the following for each i:
  - If $\mathsf{Dec}_{SK}(C'_i) \neq \perp$, then set $(x_i, r'_i) \leftarrow \mathsf{Dec}_{SK}(C'_i)$.
  - Else, if $\mathsf{RigExtract}_{SK}(C'_i, S_j) \neq \perp$ for some j, set $r'_i := r_i \cdot \mathsf{RigExtract}_{SK}(C'_i, S_j)$.
  - If both these operations fail, send CANCEL to $\mathcal{F}_{\mathsf{poll}}$ on behalf of P_{server}.

  If $\prod_i r'_i \neq \prod_i r_i$ or for some $j > k$, there is more than one i such that $\mathsf{RigExtract}_{SK}(C'_i, S_j) \neq \perp$, then send CANCEL to $\mathcal{F}_{\mathsf{poll}}$ on behalf of P_{server}. Otherwise, let σ be any permutation on $[n]$ that maps each $j > k$ to the unique i such that $\mathsf{RigExtract}_{SK}(C'_i, S_j) \neq \perp$. Send $[\text{INPUT}, x_{\sigma}(i)]$ to $\mathcal{F}_{\mathsf{poll}}$ on behalf of corrupt P_i ($i \leq k$), and then send OK to $\mathcal{F}_{\mathsf{poll}}$ on behalf of P_{server}, with σ as the permutation that $\mathcal{F}_{\mathsf{poll}}$ expects.

In this case, the primary task of $\mathcal{S}$ is to determine whether the corrupt P_{server} gives a valid list of ciphertexts to P_{client}. Applying the HCCA definition in a sequence of hybrid interactions, we see that the behavior of the real world interaction versus this simulation interaction is preserved when appropriately replacing Enc/Dec with RigEnc/RigExtract.

Note that the adversary's view is independent of $r_{k+1}, \ldots, r_n$. If $\mathsf{Dec}_{SK}(C'_i) \neq \perp$, then the corresponding r'_i value computed by the simulator is also independent of $r_{k+1}, \ldots, r_n$. Thus the only way $\prod_i r_i = \prod_i r'_i$ can be satisfied with non-negligible probability is if for each honest party P_j, exactly one i satisfies $\mathsf{RigExtract}_{SK}(C'_i, S_j) \neq \perp$. In this case, there will be exactly as many x_i's as corrupt players, and the simulator can legitimately send these to $\mathcal{F}_{\mathsf{poll}}$ as instructed (with the appropriate permutation).

Boolean OR on Encrypted Data. Using a similar technique, we can obtain a UC-secure protocol for a boolean-OR functionality. This functionality is identical to $\mathcal{F}_{\mathsf{poll}}$ except that P_{server} also gets to provide an input (say we identify P_{server} with P_0), and instead of giving $(x_{\sigma(0)}, \ldots, x_{\sigma(n)})$, it gives $\bigvee_i x_i$ as the output to P_{client}.

We can achieve this new functionality with a similar protocol — this time, using an encryption scheme that is unlinkable HCCA-secure with respect to all group operations in $\mathbb{G}^2$. P_{client} sends shares r_i to the input parties as before. The

input parties send $\mathsf{Enc}_{PK}(x_i, r_i)$ to P_{server}, where $x_i = 1$ if P_i's input is 0, and x_i is randomly chosen in $\mathbb{G}$ otherwise. Then, P_{server} rerandomizes the r_i shares as before, and also randomizes the x_i's in the following way: P_{server} multiplies each x_i by s_i such that $\prod_i s_i = 1$ if P_{server}'s input is 0, and $\prod_i s_i$ is random otherwise (P_{server} can randomize both sets of shares simultaneously using the homomorphic operation). P_{client} receives the processed ciphertexts and ensures that $\prod_i r'_i = 1$. Then if $\prod_i x'_i = 1$, it outputs 0, else it outputs 1.

We note that this approach to evaluating a boolean OR (where the induced distribution is a fixed element if the result is 0, and is random if the result is 1) has previously appeared elsewhere, e.g., [5, 6].

Relation to Voting. Our opinion polling protocol falls short of a solution for the classic election scenario in several aspects. First, in our scheme, respondents can cause the entire protocol to abort. Second, the respondents have no stake in the correctness of the results; if the pollster publishes the entire set of responses, there is no way for respondents to verify its correctness. Respondents may submit their vote accompanied by a randomly chosen nonce — this would allow a respondent to verify that his own response was included, but not that the entire set of responses is valid. Adding a publicly published nonce also allows trivial vote-selling. We finally note that an election protocol (in which all participants receive guaranteed correct results) is not possible in the plain UC model, given the impossibility results of [21].

4 Non-malleable Homomorphic Encryption for Binary Operations

In [20], it was shown that no homomorphic encryption can be completely unlinkable and also allow a group operation over the message space as a *binary* homomorphic operation — that is, an operation that multiplies two encrypted group elements. Still, the impossibility result left open the possibility of achieving a relaxation of these requirements. We consider a relaxation similar to [23]; namely, we allow the ciphertext to leak the number of operations applied to it (i.e., the depth of the circuit applied), but ideally no additional information.

Informally, we associate a *length* parameter with each ciphertext. If a length-ℓ and a length-ℓ' ciphertext are combined, then the result is a length $\ell + \ell'$ ciphertext.

Security Definition. Our formal definition is in the form of an ideal functionality in the UC framework. It is a generalization of the "homomorphic message posting" functionality presented in [20], to the case where multiple messages can be combined. The functionality, called $\mathcal{F}_{\mathbb{G}}$, is given in full detail in Figure 2. Below we explain and motivate the details of the definition.

The $\mathcal{F}_{\mathbb{G}}$ functionality allows users to post messages to each other, as on a bulletin board. The messages are stored in the functionality's memory, and are not given out except to the designated recipient. Instead, messages can be referred to using abstract *handles*, which reveal no information about the message.

The functionality keeps track of a database of records of the form $(\mathsf{handle}, \ell, m)$. Let $\mathsf{GetHandle}(args)$ be a subroutine which sends $[\text{HANDLE-REQ}, args]$ to the adversary and expects in return a string handle. If handle is previously recorded in the database, abort; otherwise, return handle.

Setup: On receiving a command [SETUP] from a party P: If a previous SETUP command has been processed, abort. Else, send $[\text{ID-REQ}, P]$ to the adversary, and expect in response a string id. Broadcast $[\text{ID-ANNOUNCE}, P, \mathsf{id}]$ to all other parties.

Dummy handles: On receiving a command $[\text{DUMMY}, \ell, \mathsf{handle}]$ from a *corrupt party only*, internally record $(\mathsf{handle}, \ell, \bot)$ and broadcast $[\text{HANDLE-ANNOUNCE}, \mathsf{handle}]$ to all parties.

Posting messages: On receiving a command $[\text{POST}, \ell, m_0, \mathsf{handle}_1, \ldots, \mathsf{handle}_k]$ from a party sender: If any handle_i is not recorded internally, or $m_0 \notin \mathbb{G}$, ignore the request. Otherwise, suppose $(\mathsf{handle}_i, \ell_i, \mathsf{msg}_i)$ is recorded for each i. If $\ell < \sum_i \ell_i$, ignore the request. Let $D = \{i \mid m_i = \bot\} \subseteq [k]$, the indices of the dummy handles. Set $m^* = m_0 * \prod_{i \notin D} m_i$, the product of known plaintexts involved.

- If $D = \emptyset$ (no dummy handles involved): If P is corrupt, set $\mathsf{handle}^* \leftarrow \mathsf{GetHandle}(\mathsf{sender}, \ell, m^*)$; otherwise let $\mathsf{handle}^* \leftarrow \mathsf{GetHandle}(\mathsf{sender}, \ell)$. Internally record $(\mathsf{handle}^*, \ell, m^*)$ and broadcast $[\text{HANDLE-ANNOUNCE}, \mathsf{handle}^*]$ to all parties.
- If $\ell > \sum_{i \in D} \ell_i$ (not entirely derived from dummy handles): If P is corrupt, set $\mathsf{handle}' \leftarrow \mathsf{GetHandle}(\mathsf{sender}, \ell', m^*)$, else set $\mathsf{handle}' \leftarrow \mathsf{GetHandle}(\mathsf{sender}, \ell')$. Internally record $(\mathsf{handle}', \ell', m^*)$.
  Set $\mathsf{handle}^* \leftarrow \mathsf{GetHandle}(\mathsf{sender}, \ell, \{\mathsf{handle}'\} \cup \{\mathsf{handle}_i \mid i \in D\})$. Internally record $(\mathsf{handle}^*, \ell, \bot)$ and send $[\text{HANDLE-ANNOUNCE}, \mathsf{handle}^*]$ to all parties.
- Otherwise (dummy handles only), Set $\mathsf{handle}^* \leftarrow \mathsf{GetHandle}(\mathsf{sender}, \ell, m_0, \{\mathsf{handle}_i \mid i \in D\})$. Internally record $(\mathsf{handle}^*, \ell, \bot)$ and send $[\text{HANDLE-ANNOUNCE}, \mathsf{handle}^*]$ to all parties.

Message reading: On receiving a command $[\text{GET}, \mathsf{handle}]$ from party P (who gave the first SETUP command): If $(\mathsf{handle}, \ell, \mathsf{msg})$ is recorded internally, send msg to P; else send $\bot$.

Fig. 2. UC ideal functionality $\mathcal{F}_{\mathbb{G}}$, parametrized by a cyclic group $\mathbb{G}$..

Following our desired intuition, users can only generate new messages in two ways (for uniformity, all handled in the same part of the functionality's code). A user can simply post a message by supplying a group element m (this is the case where $k = 0$ in the user's POST command). Alternatively, a user can provide a list of existing handles along with a group element m. If all these handles correspond to honestly-generated posts, then this has the same effect as if the user posted the product of all the corresponding messages (though note that the user does not have to know what these messages are to do this). We model the fact that handles reveal nothing about the message by letting the adversary choose the actual handle string, without knowledge of the message. The designated recipient can obtain the message by providing a handle to the

functionality. Note that there is no way (even for corrupt parties) to generate a handle derived from existing handles in a non-approved way.

However, (as in [20]) adversaries can also post *dummy handles*, which contain no message. When a user posts a derived message using such a handle, the resulting handle also contains no message. However, the adversary is also told that the handle was used in a derived POST command. The adversary also gets access to an "intermediate" handle corresponding to all the non-DUMMY handles that were combined in the POST request. Still, the adversary learns nothing about the messages corresponding to these handles. This weakness is slight and natural, since the adversary could output a ciphertext encrypted under some key unknown to the other participants. The ciphertext would be meaningless to the other parties, but the adversary could also be able to detect when someone has derived another message using it.

One may of course consider *interactive* protocols for $\mathcal{F}_{\mathbb{G}}$. However, we restrict attention to non-interactive protocols obtained via *encryption schemes* — where KeyGen implements the SETUP command, Enc and CTrans implement the POST command, and Dec implements the GET command, all in the natural ways.

The Construction. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CTrans})$ be an unlinkable HCCA-secure scheme, whose message space is $\mathbb{G}^2$ for a cyclic group $\mathbb{G}$, and whose allowed (unary) transformations are all group operations in $\mathbb{G}^2$. We suppose the CTrans operation accepts arguments as $\mathsf{CTrans}(C, (r, s))$, where $r, s \in \mathbb{G}$ specify the transformation $(\alpha, \beta) \mapsto (r\alpha, s\beta)$. We abbreviate the $\mathsf{CTrans}(C, (r, s))$ operation as "$(r, s) * C$". Thus $(r, s) * \mathsf{Enc}_{PK}(\alpha, \beta)$ is indistinguishable from $\mathsf{Enc}_{PK}(r\alpha, s\beta)$, in the sense of the unlinkability definition.

The new scheme $\mathcal{E}^*$ is given by the following algorithms:

Key generation (KeyGen^*) Same as KeyGen.

Encryption (Enc^*) To encrypt an element $m \in \mathbb{G}$ in a length-ℓ ciphertext, output

$$C = \Big(\mathsf{Enc}_{PK}(\alpha_1, \beta_1), \ldots, \mathsf{Enc}_{PK}(\alpha_\ell, \beta_\ell)\Big)$$

where α_i, β_i are randomly chosen in $\mathbb{G}$ subject to the constraint $\prod_i \alpha_i = \prod_i \beta_i = m$.

Decryption (Dec^*) To decrypt a ciphertext $C = (C_1, \ldots, C_\ell)$, decrypt each C_i to get (α_i, β_i). If any decryption returns $\perp$, or if $\prod_i \alpha_i \neq \prod_i \beta_i$, output $\perp$. Else output $\prod_i \alpha_i$.

Transformation operation (CTrans^*) To "multiply" two given ciphertexts $C = (C_1, \ldots, C_\ell)$ and $C' = (C_1, \ldots, C_{\ell'})$, output a random permutation of:

$$\Big((r_1, s_1) * C_1, \ldots, (r_\ell, s_\ell) * C_\ell, (r_{\ell+1}, s_{\ell+1}) * C'_1, \ldots, (r_{\ell+\ell'}, s_{\ell+\ell'}) * C'_{\ell'}\Big)$$

where r_i, s_i are randomly chosen in $\mathbb{G}$ subject to $\prod_i r_i = \prod_i s_i = 1$

To "multiply" a single given ciphertext $C = (C_1, \ldots, C_\ell)$ by a given known group element $R \in \mathbb{G}$ (without increasing the ciphertext length), output:

$$\Big((r_1, s_1) * C_1, \ldots, (r_\ell, s_\ell) * C_\ell\Big)$$

where r_i, s_i are randomly chosen in $\mathbb{G}$ subject to $\prod_i r_i = \prod_i s_i = R$.

We note that the syntax of CTrans^* can be naturally extended to support multiplying several ciphertexts and/or a known group element at once, simply by composing the operations described above.

Theorem 3. *If $\mathcal{E}$ is unlinkable and HCCA-secure with respect to $\mathbb{G}^2$, where $|\mathbb{G}|$ is superpolynomial in the security parameter, then $\mathcal{E}^*$ (as described above) is a secure realization of $\mathcal{F}_\mathbb{G}$, with respect to static corruptions.*

Proof. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CTrans})$ be the unlinkable HCCA-secure scheme used as the main component in our construction, and let RigEnc and $\mathsf{RigExtract}$ be the procedures guaranteed by HCCA security.

We proceed by constructing an ideal-world simulator for any arbitrary real-world adversary $\mathcal{A}$. The simulator $\mathcal{S}$ is constructed by considering a sequence of hybrid functionalities that culminate in $\mathcal{F}_\mathbb{G}$. These hybrids differ from $\mathcal{F}_\mathbb{G}$ only in how much they reveal in their HANDLE-REQ requests to the adversary.

Correctness. Note that $\mathcal{F}_\mathbb{G}$ only makes two kinds of HANDLE-REQ requests: those containing a lone message, and those containing a list of handles.

Let $\mathcal{F}_1$ be the functionality that behaves exactly as $\mathcal{F}_\mathbb{G}$, except that every time it sends a HANDLE-REQ to the simulator, it also includes the entire party's input that triggered the HANDLE-REQ. Define $\mathcal{S}_1$ to be the simulator that internally runs the adversary $\mathcal{A}$, and does the following:

- When $\mathcal{F}_1$ gives $(\text{ID-REQ}, P)$ to $\mathcal{S}_1$, it generates a key pair $(PK, SK) \leftarrow \mathsf{KeyGen}$ and responds with PK. It simulates to $\mathcal{A}$ that party P broadcast PK.
- When $\mathcal{F}_1$ gives a HANDLE-REQ to $\mathcal{S}_1$, it generates the handle appropriately — with either Enc^*_{PK} or CTrans^* on an existing handle, depending on the party's original command which is included in the HANDLE-REQ. It simulates to $\mathcal{A}$ that the appropriate party output the handle.
- When $\mathcal{A}$ broadcasts a length-ℓ ciphertext C, $\mathcal{S}_1$ tries to decrypt it with Dec^*_{SK}. If it decrypts (say, to m), then $\mathcal{S}_1$ sends a (POST, ℓ, m) command to $\mathcal{F}_1$ and later gives C as the handle; else it sends (DUMMY, ℓ, C).

$\mathcal{S}_1$ exactly simulates the honest parties' behavior in the real world interaction. By the correctness properties of $\mathcal{E}^*$, the outputs of the honest ideal-world parties match that of the real world, except with negligible probability; thus, $\text{REAL}^{\mathcal{E}^*}_{\mathcal{Z},\mathcal{A}} \approx \text{IDEAL}^{\mathcal{F}_1}_{\mathcal{Z},\mathcal{S}_1}$ for all environments $\mathcal{Z}$.

Unlinkability. Let $\mathcal{F}_2$ be exactly like $\mathcal{F}_1$, except for the following change: For requests of the form $[\text{HANDLE-REQ}, \mathsf{sender}, \ell, m]$, $\mathcal{F}_2$ does not send the handles that caused this request. That is, whereas $\mathcal{F}_1$ would tell the simulator that the handle is being requested for a POST command combining some non-dummy handles, $\mathcal{F}_2$ would instead act like sender had sent $[\text{POST}, \ell, m]$ (that this is closer to what $\mathcal{F}_\mathbb{G}$ does; internally behaving identically for such requests). Let $\mathcal{S}_2 = \mathcal{S}_1$, since $\mathcal{F}_1$ is only sending one fewer type of HANDLE-REQ to the simulator.

By a standard hybrid argument, we can see that $\text{IDEAL}^{\mathcal{F}_1}_{\mathcal{Z},\mathcal{S}_1} \approx \text{IDEAL}^{\mathcal{F}_2}_{\mathcal{Z},\mathcal{S}_2}$ for all environments $\mathcal{Z}$. The hybrids are over the number of POST requests affected by this change. Consecutive hybrids differ by whether a single handle was generated by Enc^* or by CTrans^*. The only handles that are affected here are non-DUMMY handles, and thus ciphertexts which decrypt successfully under SK. Thus distinguishing between consecutive hybrids can be reduced to succeeding in the unlinkability experiment (by further hybridizing over the individual Enc ciphertext components).

HCCA. If the owner P of the functionality is corrupt, then $\mathcal{S}_2$ is already a suitable simulator for $\mathcal{F}_{\mathbb{G}}$, and we can stop at this point.

Otherwise, the difference between $\mathcal{F}_{\mathbb{G}}$ and $\mathcal{F}_2$ is that $\mathcal{F}_{\mathbb{G}}$ does not reveal the message in certain HANDLE-REQ requests. Namely, those in which the simulator receives $[\text{HANDLE-REQ}, \mathsf{sender}, \ell]$.

Let $\mathcal{S}_3$ be exactly like $\mathcal{S}_2$, except for the following changes: Each time $\mathcal{S}_2$ would generate a ciphertext component via $\mathsf{Enc}_{PK}(\alpha, \beta)$, $\mathcal{S}_3$ instead generates it with RigEnc_{PK}. It keeps track of the auxiliary information S and records (S, α, β) internally. Also, whenever $\mathcal{S}_2$ would decrypt a ciphertext component using Dec_{SK}, $\mathcal{S}_3$ instead decrypts it via:

$$D(C) = \begin{cases} (r\alpha, s\beta) & \text{if any } (S, \alpha, \beta) \text{ is recorded such that } (r, s) \leftarrow \mathsf{RigExtract}_{SK}(C, S) \\ \mathsf{Dec}_{SK}(C) & \text{otherwise} \end{cases}$$

By a straight-forward hybrid argument (where distinguishing between consecutive hybrids reduces to distinguishing in one execution of the HCCA experiment), we have that $\text{IDEAL}^{\mathcal{F}_2}_{\mathcal{Z},\mathcal{S}_2} \approx \text{IDEAL}^{\mathcal{F}_2}_{\mathcal{Z},\mathcal{S}_3}$ for all environments $\mathcal{Z}$.

Suppose the internal records (S, α, β) are labeled as (S_j, α_j, β_j) for $j \geq 1$. Now for each HANDLE-REQ request q sent to $\mathcal{S}_3$, we define J_q to be the set of indices j such that (S_j, α_j, β_j) was generated as a result of servicing request q.

Each α, β is chosen randomly in $\mathbb{G}$, subject to a constraint on some of their products, as prescribed by Enc^* and CTrans^*. However, the ciphertexts given to the adversary are generated by RigEnc_{PK}, and thus independent of these random choices. In fact, the entire adversary's view is (essentially) independent of the random choices of α, β, subject to $\prod_{j \in J_q} \alpha_j/\beta_j$ being fixed (we pessimistically assume that $\mathcal{A}$ knows this fixed value for each q). Put another way, $\prod_{j \in J'}(\alpha_j/\beta_j)$ is uniformly distributed for a multiset J' if and only if for all q, all elements of J_q have the same multiplicity in J'.

We now examine when a ciphertext given by the adversary is successfully decrypted by the simulator (and thus given to the functionality as a POST instead of as a DUMMY handle).

Given a ciphertext (sequence of HCCA ciphertexts) $C = (C_1, \ldots, C_\ell)$, $\mathcal{S}_3$ first decrypts each C_i to obtain $(\alpha_i, \beta_i) = D(C_i)$. The overall decryption succeeds if $\prod_i (\alpha_i/\beta_i) = 1$. Let J' be the multiset of indices j such that $\perp \neq \mathsf{RigExtract}_{SK}(C_i, S_j)$, with multiplicity for each i where this holds. The decryption constraint above is uniformly distributed (and thus equality holds only

with negligible probability) unless all elements of J_q have the same multiplicity in J'. However, when all elements of J_q have the same multiplicity in J', we may cancel all the α_j/β_j terms in the constraint. What remains are terms of the form α_i/β_i, where $(\alpha_i, \beta_i) \leftarrow \mathsf{Dec}_{SK}(C_i)$, and terms of r_i/s_i, where $(r_i, s_i) \leftarrow \mathsf{RigExtract}_{SK}(C_i, S_j)$. The ciphertext then decrypts successfully if and only if the constraint holds with respect to these remaining terms.

Thus, we can consider a simulator $\mathcal{S}_4$ which behaves just like $\mathcal{S}_3$, except that when $\mathcal{A}$ outputs a ciphertext $C = (C_1, \ldots, C_\ell)$, it processes it as follows:

- If some C_i is such that $D(C_i) = \perp$, the ciphertext is invalid; send $[\text{DUMMY}, C]$ to the functionality.
- Define J' as above. If for some q, the elements of J_q do not all have the same multiplicity in J', the ciphertext is invalid; send $[\text{DUMMY}, C]$ to the functionality.
- Let I be the set of indices such that $\perp \neq (\alpha_i, \beta_i) \leftarrow \mathsf{Dec}_{SK}(C_i)$. If $\prod_{i \in I}(\alpha_i/\beta_i) \neq 1$, then the ciphertext is invalid; send $[\text{DUMMY}, C]$ to the functionality.
- Let $(r_i, s_i) \leftarrow \mathsf{RigExtract}_{SK}(C_i, S_j)$ for each $i \notin I$, If $\prod_{i \notin I}(r_i/s_i) \neq 1$, then the ciphertext is invalid; send $[\text{DUMMY}, C]$ to the functionality.
- Otherwise, send $[\text{POST}, \ell, m_0, \{\mathsf{handle}_j \mid j \in J'\}]$ to the functionality, where $m_0 = \prod_{i \in I} \alpha_i \prod_{i \notin I} r_i$.

Except with negligible probability, $\mathcal{S}_4$ interacts identically with the functionality as $\mathcal{S}_3$. However, note that $\mathcal{S}_4$ does not actually look at the α_j, β_j values that are recorded for each call to RigEnc. Thus $\mathcal{S}_4$ can be successfully implemented even if the functionality does not reveal m in messages of the form $[\text{HANDLE-REQ}, \mathsf{sender}, \ell, m]$. Therefore $\mathcal{S}_4$ is a suitable simulator for $\mathcal{F}_{\mathbb{G}}$ itself, and $\text{IDEAL}^{\mathcal{F}_2}_{\mathcal{Z},\mathcal{S}_3} \approx \text{IDEAL}^{\mathcal{F}_{\mathbb{G}}}_{\mathcal{Z},\mathcal{S}_4}$ for all environments $\mathcal{Z}$.

Acknowledgments

We would like to thank Josh Benaloh and the anonymous referees for suggesting helpful improvements.

References

1. M. Abadi and J. Feigenbaum. Secure circuit evaluation. *J. Cryptology*, 2(1):1–12, 1990.
2. M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. *J. Comput. Syst. Sci.*, 39(1):21–50, 1989.
3. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, pages 119–135, 2001. LNCS No. 2045.
4. D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.
5. D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In J. Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

6. A. Broadbent and A. Tapp. Information-theoretic security without an honest majority. In K. Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 410–426. Springer, 2007.
7. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005.
8. Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In J. Ioannidis, A. D. Keromytis, and M. Yung, editors, *ACNS*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455, 2005.
9. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 4(2), February 1981.
10. B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. TR CS0917, Department of Computer Science, Technion, 1997.
11. Y. Desmedt. Computer security by redefining what a computer is. In *NSPW '92-93: Proceedings on the 1992-1993 workshop on New security paradigms*, pages 160–166, New York, NY, USA, 1993. ACM Press.
12. J. Feigenbaum. Encrypting problem instances: Or ..., can you take advantage of someone without having to trust him? In H. C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 477–488. Springer, 1985.
13. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In J. Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.
14. E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.
15. P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. In M. Jakobsson, M. Yung, and J. Zhou, editors, *ACNS*, volume 3089 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2004.
16. J. Groth. A verifiable secret shuffle of homomorphic encryptions. In Y. Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2003.
17. Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.
18. W. Ogata and K. Kurosawa. Oblivious keyword search. *J. Complexity*, 20(2-3):356–371, 2004.
19. D. J. Park, K. Kim, and P. J. Lee. Public key encryption with conjunctive field keyword search. In C. H. Lim and M. Yung, editors, *WISA*, volume 3325 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2004.
20. M. Prabhakaran and M. Rosulek. Homomorphic encryption with CCA security. To appear in ICALP '08. Full version available from `http://eprint.iacr.org/2008/079`, 2008.
21. M. Prabhakaran and M. Rosulek. Homomorphic encryption with CCA security. In *ICALP*, 2008. Full version available from `http://eprint.iacr.org/2008/079`.
22. R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of secure computation (Workshop, Georgia Inst. Tech., Atlanta, Ga., 1977)*, pages 169–179. Academic, New York, 1978.
23. T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *FOCS*, pages 554–567, 1999.
24. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

A Security Definitions for Non-Malleable Homomorphic Encryption

The formal definitions in this section are summarized from [20] for reference:

HCCA Security. The main security definition, called *Homomorphic-CCA (HCCA)* security, formalizes the intuition that a homomorphic encryption scheme is "non-malleable except for a certain set of operations." The complete security experiment is given in Figure 3, and we give an overview and motivation below.

Definition 1. *A homomorphic encryption scheme is* Homomorphic-CCA (HCCA) secure *with respect to $\mathcal{T}$ if there are PPT algorithms* RigEnc *and* $\mathsf{RigExtract}$*, where the range of* $\mathsf{RigExtract}$ *is $\mathcal{T} \cup \{\perp\}$, and such that for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the IND-HCCA experiment (Figure 3) is negligible.*

When $b = 0$ in the experiment, the adversary simply receives an encryption of his chosen plaintext msg^*, and gets access to an unrestricted decryption oracle. However, when $b = 1$ in the experiment, instead of an encryption of msg^*, the adversary receives a "rigged" ciphertext generated by RigEnc, without knowledge of msg^*. Such a rigged ciphertext need not encode any actual message, so if the adversary asks for it (or any of its derivatives via the homomorphic operations) to be decrypted, the decryption oracle's response must be compensated in some way, or else it would be easy to distinguish the $b = 0$ from $b = 1$ scenarios. For this purpose, the RigEnc procedure also produces some (secret) extra state information, which makes it possible to identify (via the $\mathsf{RigExtract}$ procedure) all ciphertexts derived from that particular rigged ciphertext, as well as *how* they were derived. So in the $b = 1$ scenario, the decryption oracle first uses $\mathsf{RigExtract}$ to check whether the given ciphertext was derived via a homomorphic operation of the scheme, and if so, compensates in its response. For example, if the query ciphertext was derived by applying the T transformation, then the decryption oracle should respond with $T(\mathsf{msg}^*)$, to mimic the $b = 0$ case.

It is easily seen that if it is feasible for an adversary to modify an encryption of $\mathsf{Enc}(\mathsf{msg})$ into a related encryption $\mathsf{Enc}(T(\mathsf{msg}))$, but $\mathsf{RigExtract}$ never outputs T, then there is a way for an adversary to distinguish between $b = 0$ and $b = 1$ in the experiment. Thus by restricting the range of the $\mathsf{RigExtract}$ procedure in the security definition, we limit the feasible malleability of the scheme.

Finally, because $\mathsf{RigExtract}$ uses the private key, as well as secret auxiliary information from RigEnc, we should provide an oracle for these procedures. We do so in a "guarded" way that keeps the auxiliary shared information hidden from the adversary in the experiment.

Unlinkability. The second security definition, called *unlinkability*, formalizes of the natural requirement that a ciphertext hides not only its plaintext, but also its "history" — i.e., whether it was generated as a normal Enc, or by applying the homomorphic operations to some other ciphertext.

We note that the definition is more than just a correctness property, as it involves the behavior of the scheme's algorithms on maliciously-crafted ciphertexts.

Setup: Pick $(PK, SK) \leftarrow \mathsf{KeyGen}$ and give PK to $\mathcal{A}$.
Phase I: $\mathcal{A}$ gets access to the $\mathsf{Dec}_{SK}(\cdot)$ oracle and the following two "guarded" RigEnc and RigExtract oracles:

$$\mathsf{GRigEnc}_{PK}() = \zeta_i, \text{ where } (\zeta_i, S_i) \leftarrow \mathsf{RigEnc}_{PK}, \text{ when called for the } i\text{th time}$$
$$\mathsf{GRigExtract}_{SK}(\zeta, i) = \mathsf{RigExtract}_{SK}(\zeta, S_i)$$

Challenge: $\mathcal{A}$ outputs a plaintext msg^*. We privately flip a coin $b \leftarrow \{0,1\}$. If $b = 0$, we compute $\zeta^* \leftarrow \mathsf{Enc}_{PK}(\mathsf{msg}^*)$. If $b = 1$, we compute $(\zeta^*, S^*) \leftarrow \mathsf{RigEnc}_{PK}$. In both cases, we give ζ^* to $\mathcal{A}$.
Phase II: $\mathcal{A}$ gets access to the same GRigEnc and GRigExtract oracles as in Phase I, as well as a "rigged" version of the decryption oracle RigDec. When $b = 0$, RigDec is simply the normal decryption oracle $\mathsf{Dec}_{SK}(\cdot)$. When $b = 1$, RigDec is implemented as follows:

$$\mathsf{RigDec}_{SK}(\zeta) = \begin{cases} T(\mathsf{msg}^*) & \text{if } \perp \neq T \leftarrow \mathsf{RigExtract}_{SK}(\zeta, S^*) \\ \mathsf{Dec}_{SK}(\zeta) & \text{otherwise} \end{cases}.$$

Output: $\mathcal{A}$ outputs a bit b'. The *advantage* of $\mathcal{A}$ is $\Pr[b' = b] - \frac{1}{2}$.

Fig. 3. IND-HCCA security experiment, parametrized by $\mathcal{T}$

The security experiment also includes a decryption oracle, making it applicable even to adversaries with chosen-ciphertext attack capabilities.

Definition 2. *A homomorphic encryption scheme is* unlinkably homomorphic *with respect to $\mathcal{T}$ if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the unlinkability experiment (Figure 4) is negligible.*

Setup: Pick $(PK, SK) \leftarrow \mathsf{KeyGen}$ and give PK to $\mathcal{A}$.
Phase I: $\mathcal{A}$ is given access to the decryption oracle $\mathsf{Dec}_{SK}(\cdot)$.
Challenge: Flip a coin $b \leftarrow \{0,1\}$. $\mathcal{A}$ outputs a ciphertext ζ and a transformation $T \in \mathcal{T}$. If $\mathsf{Dec}_{SK}(\zeta) = \perp$, do nothing. Else give ζ^* to $\mathcal{A}$ where

$$\zeta^* \leftarrow \begin{cases} \mathsf{Enc}_{PK}(T(\mathsf{Dec}_{SK}(\zeta))) & \text{if } b = 0 \\ \mathsf{CTrans}(\zeta, T) & \text{if } b = 1 \end{cases}.$$

Phase II: $\mathcal{A}$ is given access to the decryption oracle $\mathsf{Dec}_{SK}(\cdot)$.
Output: $\mathcal{A}$ outputs a bit b'. The *advantage* of $\mathcal{A}$ is $\Pr[b' = b] - \frac{1}{2}$.

Fig. 4. Unlinkability security experiment, parametrized by $\mathcal{T}$