Garbled Circuits via Structured Encryption

Seny Kamara
Microsoft Research

Lei Wei [1]
UNC-Chapel Hill

Abstract. The garbled circuit technique transforms a circuit in such a way that it can be evaluated on encrypted inputs. Garbled circuits were originally introduced by Yao (FOCS '86) for the purpose of secure two-party computation but have since found many applications.
In this work, we consider the problem of designing *special-purpose* garbled circuits, which are garbled circuits that handle only a specific class of functionalities. Special-purpose constructions are usually smaller than general-purpose ones and lead to more efficient two-party protocols.
We propose a design framework for constructing special-purpose garbled circuits based on structured encryption schemes, which are encryption schemes that encrypt data structures in such a way that they can be queried through the use of a token. Using our framework, we show how to design more efficient garbled circuits for several graph-based functionalities (with applications to online social network analysis), Boolean circuits, deterministic finite automata, and branching programs.

1 Introduction

Yao's garbled circuit technique transforms circuits in such a way that they can be evaluated on encrypted inputs. While garbled circuits were originally introduced for the purpose of two-party secure function evaluation (SFE) [19], they have since found many applications, some of which include the design of homomorphic encryption schemes, one-time programs, circular-secure encryption, non-interactive verifiable computation, functional encryption, and single-server-aided SFE.

At a high level, the garbled circuit technique consists of: (1) a garbling procedure that transforms a circuit C that computes a function f, and a set of inputs $\mathbf{x} = (x_1, \ldots, x_n)$ into a garbled circuit $\widetilde{\mathrm{C}}$ and an encoded input $\widetilde{\mathbf{x}} = (\widetilde{x}_1, \ldots, \widetilde{x}_n)$; (2) an evaluation procedure that computes a garbled output $\widetilde{\mathbf{y}}$ given $\widetilde{\mathrm{C}}$ and $\widetilde{\mathbf{x}}$; and (3) a decoding procedure that, given $\widetilde{\mathbf{y}}$ and a set of decoding keys $\mathbf{dk}$ returns $f(x)$. The main security property provided by garbled circuits is *input privacy*, which guarantees that, given $(\widetilde{\mathrm{C}}, \widetilde{\mathbf{x}}, \mathbf{dk})$, no information about $\mathbf{x}$ is revealed by the garbled circuit evaluation beyond what can be inferred from $f(\mathbf{x})$. As shown by Yao, combining garbled circuits with oblivious transfer results in constant-round two-party SFE secure against semi-honest adversaries.

The importance of the garbled circuit technique in cryptography can be attributed to several factors, including its security properties, its relative efficiency and, most importantly, its generality. In fact, like fully-homomorphic encryption, garbled circuits are one of the few general-purpose primitives in cryptography.

[1] Work done while at Microsoft Research.

While generality is crucial for establishing completeness theorems and for understanding the power of cryptographic techniques, it is well-known that it often comes at the price of efficiency. In fact, it is common for special-purpose constructions (i.e., constructions that handle only a a sub-class of functionalities) to be more efficient than general-purpose constructions.

Our contributions. In this work, we consider the problem of designing special-purpose garbling schemes. Given the importance of garbled circuits and the efficiency improvements enjoyed by special-purpose constructions, this is a natural and well-motivated problem. We make the following contributions.

We introduce a general framework for designing special-purpose garbling schemes. Our framework is based on a connection between garbled circuits and the notion of *structured encryption* [8] which is a generalization of index-based searchable symmetric encryption (SSE) [18,10,7,9]. Roughly speaking, a structured encryption scheme encrypts a data structure in such a way that it can be queried through the use of a query-specific token that does not reveal information about the query. Our approach essentially reduces the problem of designing special-purpose garbled circuits to the problem of designing structured encryption schemes. Consequently, improvements in either the efficiency or functionality of structured encryption can lead to similar improvements in the design of special-purpose two-party protocols in the semi-honest model and other cryptographic primitives that rely on input-private garbled circuits.

While our main contributions are conceptual, we demonstrate the utility of our approach by constructing special-purpose garbling schemes for several useful functionalities. For example, using our framework with the structured encryption schemes of [8], we get special-purpose garbling schemes (and therefore two-party protocols) for several graph-based functionalities that have applications to online social networks. In addition, in the full version of this work we use our framework to construct garbling schemes for other functionalities like branching programs (BP), deterministic finite automata (DFA) and even Boolean circuits. In all cases, the garbled circuits resulting from our approach are more efficient (i.e., either smaller or with faster evaluation) than the garbled circuits that would result from applying Yao's general-purpose construction.

The main building block we need to handle DFAs, BPs and Boolean circuits is a matrix encryption scheme that supports lookups, i.e., a structured encryption scheme that encrypts matrices in such a way that a location (i, j) can be queried using a token. While such a scheme is described in [8], that particular construction is not appropriate for our purposes. The problem is that the scheme from [8] is only 1-dimensional in the sense that it generates a single token for a location (i, j) in the matrix. For our purposes, however, we need a 2-dimensional scheme that generates two independent tokens, i.e., one for i and one for j that can be combined to lookup location (i, j). We show how to construct such a scheme based on the 1-dimensional construction of [8] and pseudo-random synthesizers [17].

1.1 Background on Structured Encryption

Several variants of structured encryption were described in [8] but for our purposes we need the *structure-only* variant which only encrypts data structures as opposed to the standard variant which also encrypts messages. A structured encryption scheme is a tuple of four polynomial-time algorithms $\mathsf{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Query_e})$ such that Gen is a probabilistic algorithm that takes as input a security parameter k and outputs a private key K. Let $\mathscr{T}$ be an abstract data type that maps queries q from a query space $\mathcal{Q}$ to an answer a from a response space $\mathcal{R}$. Enc is a probabilistic algorithm that takes as input a key K, a data structure $\delta \in \mathscr{T}$ and outputs an encrypted data structure γ. Token is a (possibly probabilistic) algorithm that takes as input a private key K and a query $q \in \mathcal{Q}$ and outputs a token τ. $\mathsf{Query_e}$ is a deterministic algorithm that takes as input an encrypted data structure γ and a token τ and outputs an answer $a \in \mathcal{R}$. Informally, a structured encryption scheme is secure against chosen-query attacks (CQA1) if no useful information about q and δ can be recovered from γ and τ beyond what can be deduced from a. We say that a structured encryption scheme is secure against *adaptive* chosen-query attacks (CQA2) if this holds even when queries are made adaptively (i.e., as a function of the encrypted data structure γ and the results of previous queries and tokens).

As a concrete example, consider a graph encryption scheme $\mathsf{Graph} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Neigh}_e)$ that supports neighbor queries (we refer the reader to [8] for a concrete construction). With such a scheme one can encrypt the edges E of a graph $G = (V, E)$ by computing $\gamma \leftarrow \mathsf{Enc}(K, E)$. A token for a vertex $v \in V$ can be created as $\tau \leftarrow \mathsf{Token}(K, v)$ and the neighbors of v, denoted $\Gamma(v)$, can be recovered by computing $\mathsf{Neigh}_e(\gamma, \tau)$.

Associative structured encryption. For our purposes, we need *associative* structured encryption schemes which allow one to associate arbitrary strings to each output. So, with respect to our previous example, an associative graph encryption scheme supporting neighbor queries would: (1) allow the encryptor to associate arbitrary strings to each vertex of the graph during the encryption step; and (2) reveal these strings whenever the associated vertex is in $\Gamma(v)$. More precisely, in addition to the secret key sk and the edges E, the Enc algorithm would also take as input a set of strings $(s_{v_1}, \ldots, s_{v_{|V|}})$, where s_{v_i} is associated with vertex v_i. Then, the algorithm Neigh_e would return, in addition to $\Gamma(v)$, the set $\{s_w\}_{w \in \Gamma(v)}$.

Due to space restrictions, we refer the reader to [8] for formal definitions of (associative) structured encryption and of the relevant security definitions.

1.2 Overview of our Framework

At a high level, our framework consists of two steps. In the first step, the function f is represented as a *structured circuit* which is a circuit-like computational model where each gate g can query a data structure δ and where the input

and output wires of g carry queries for the structures of g and g's descendent, respectively. Our notion of structured circuits is reminiscent of Naor and Nissim's circuits with lookup tables [16] though, in our setting, the contents of the data structure cannot be set during computation. In the second step, at a very high level, the structured circuit is garbled by encrypting each data structure δ with an appropriate structured encryption scheme. These encrypted structures are viewed as the garbled gates and the tokens used to query them are viewed as the encoded wire values.

Note that the functionality and security properties needed to construct a garbled gate are precisely what is provided by associative structured encryption schemes. Indeed, a garbled gate must: (1) privately store the encodings for its outgoing wires; (2) reveal those encodings when presented with encodings for its input wires (according to the operation implemented by the gate); and (3) not reveal anything about a wire value given only its encoding. Similarly, an associative structured encryption scheme encrypts a data structure in such a way that: (1) arbitrary strings can be stored in the encrypted structure and associated with any answer; (2) these strings are only revealed when presented with an appropriate query token; and (3) no information is revealed about the query from the token.

2 Related Work

Garbled circuits. Garbled circuits were introduced by Yao in his seminal work on SFE [19]. Since, they have found many additional applications as discussed in Section 1. Due to their wide applicability, several garbling techniques have been introduced over the years. Recently, Applebaum, Ishai and Kushilevitz proposed the first garbling scheme for arithmetic circuits [1].

Formalizations of garbled circuits have been proposed in the past. The most notable is the notion of randomized encodings (RE), which was introduced by Ishai and Kushilevitz [12,13]. While REs have found many applications in cryptography and even in complexity theory, the RE abstraction is not appropriate for certain applications. Recently, Bellare, Hoang and Rogaway [4,3] provided a formal treatment of garbled circuits. The formalization we use here is similar to that of [4,3] but does not capture function privacy (which, intuitively, guarantees that a garbled circuit does not reveal information about its functionality) since we are mostly concerned here with applications to two-party computation (as opposed to private function evaluation).

Special-purpose garbled circuits. In addition to the general-purpose constructions described above, several works in the past have proposed two-party protocols for various classes of functions that (sometimes implicitly) relied on special-purpose garbled circuits. Some examples are [14,6,2], which construct efficient two-party protocols for evaluating ordered binary decision diagrams (OBDDS); and [15] which gives an efficient protocol for evaluating DFAs. All these protocols can be viewed as a combination of a special-purpose garbling scheme with OT, just as

Yao's general-purpose two-party protocol is a combination of a general-purpose garbling scheme with OT.

Structured encryption. Structured encryption was introduced in [8] as a generalization of the notion of a secure index considered by Song, Wagner and Perrig in [18] and Goh in [10] for the purpose of building searchable symmetric encryption (SSE) schemes. SSE was first considered explicitly in [18].

3 Preliminaries

We use oracles in some of our definitions for conciseness. In each case, these oracles only allow the adversary to make a *single* query. To stress this, we will often say that an oracle is a single-query oracle.

An *abstract data type* is a collection of objects together with a set of operations defined on those objects. We recall the formalization of an abstract data type given in [8]. Formally, a data type $\mathscr{T}$ is defined by a universe $\mathcal{U} = \{\mathcal{U}_k\}_{k\in\mathbb{N}}$ and an operation $\mathsf{Query} : \mathcal{U} \times \mathcal{Q} \to \mathcal{R}$, where $\mathcal{Q} = \{\mathcal{Q}_k\}_{k\in\mathbb{N}}$ is the operation's query space and $\mathcal{R} = \{\mathcal{R}_k\}_{k\in\mathbb{N}}$ is its response space. The universe, query and response spaces are ensembles of finite sets indexed by the security parameter k. We assume that the universe is a totally ordered set and that the response space includes special elements $\perp$ and ϵ denoting failure and the empty string, respectively. Given a data structure δ we sometimes write $\mathscr{T}(\delta)$ to refer to its type.

4 Syntactic and Security Definitions

A garbling scheme Garb consists of four algorithms $(\mathsf{Grb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$. The algorithm Grb is used to garble a circuit and to generate a secret key sk and a set of decoding keys $\mathbf{dk}$. The algorithm Enc is used with the secret key to encode inputs, and the Eval algorithm is used to evaluate a garbled circuit on a set of encoded inputs. Evaluation results in an encoded output which can be decoded into the real output using the decoding algorithm Dec and an appropriate subset of the decoding keys.

Definition 1 (Garbling scheme). *A garbling scheme* $\mathsf{Garb} = (\mathsf{Grb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *consists of four polynomial-time algorithms that work as follows:*

- $(\widetilde{\mathrm{C}}, \mathbf{dk}, \mathrm{sk}) \leftarrow \mathsf{Grb}(1^k, \mathrm{C})$: *is a probabilistic algorithm that takes as input a circuit* C *with* n *inputs and* ℓ *outputs and returns a garbled circuit* $\widetilde{\mathrm{C}}$*, a set of decoding keys* $\mathbf{dk} = (\mathrm{dk}_1, \ldots, \mathrm{dk}_\ell)$ *and a secret key* sk.
- $\widetilde{x} := \mathsf{Enc}(\mathrm{sk}, x)$: *is a deterministic algorithm that takes as input a secret key* sk*, an input* x *and returns an encoded input* $\widetilde{x}$*. We sometimes write* $\widetilde{\mathbf{x}} := \mathsf{Enc}(\mathrm{sk}, \mathbf{x})$ *to denote the algorithm that takes multiple inputs* $\mathbf{x} = (x_1, \ldots, x_n)$*, runs* $\mathsf{Enc}(\mathrm{sk}, \cdot)$ *on each* x_i *and returns the garbled inputs* $\widetilde{x}_1$ *through* $\widetilde{x}_n$*.*

- $\widetilde{\mathbf{y}} := \mathsf{Eval}(\widetilde{\mathrm{C}}, \widetilde{\mathbf{x}})$: *is a deterministic algorithm that takes as input a garbled circuit* $\widetilde{\mathrm{C}}$ *and encoded inputs* $\widetilde{\mathbf{x}}$ *and returns encoded outputs* $\widetilde{\mathbf{y}}$.
- $\{\bot, y_i\} := \mathsf{Dec}(\mathrm{dk}_i, \widetilde{y}_i)$: *is a deterministic algorithm that takes as input a decoding key* dk_i *and an encoded output* $\widetilde{y}_i$ *and returns either the failure symbol* $\bot$ *or an output* y_i. *We sometimes write* $\{\bot, \mathbf{y}\} := \mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{y}})$ *to denote the algorithm that takes multiple garbled outputs* $\widetilde{\mathbf{y}} = (\widetilde{y}_1, \ldots, \widetilde{y}_\ell)$, *runs* $\mathsf{Dec}(\mathrm{dk}_i, \cdot)$ *on each* $\widetilde{y}_i$ *and returns the outputs* y_1 *through* y_ℓ.

We say that Garb *is correct if for all* $k \in \mathbb{N}$, *for all polynomial-size circuits* C, *for all inputs* $\mathbf{x}$ *for in the domain of* C, *for all* $(\widetilde{\mathrm{C}}, \mathbf{dk}, \mathrm{sk})$ *output by* $\mathsf{Grb}(1^k, \mathrm{C})$, *for* $\widetilde{\mathbf{x}} := \mathsf{Enc}(\mathrm{sk}, \mathbf{x})$ *and* $\widetilde{\mathbf{y}} := \mathsf{Eval}\big(\widetilde{\mathrm{C}}, \widetilde{\mathbf{x}}\big)$ *and for all* $i \in [\ell]$, $\mathsf{Dec}\big(\mathrm{dk}_i, \widetilde{y}_i\big) = y_i$.

Non-adaptive input privacy. Most applications of garbled circuits rely on a simple notion of security that guarantees that a garbled circuit $\widetilde{\mathrm{C}}$ together with encoded inputs $\widetilde{\mathbf{x}}$ and the decoding keys $\mathbf{dk}$ reveal at most $f(\mathbf{x})$. The following simulation-based definition guarantees that the garbled circuit, the encoded inputs and the decoding keys are all simulatable given the result of the computation. Intuitively, this implies that for some set of inputs $\mathbf{x}$, an efficient adversary that holds $(\widetilde{\mathrm{C}}, \widetilde{\mathbf{x}}, \mathbf{dk})$ will not learn anything beyond $f(\mathbf{x})$.

Definition 2 (Sim1-security). *A garbling scheme* $\mathsf{Garb} = (\mathsf{Grb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* Sim1*-secure with respect to a circuit* C *if, for all polynomial-size adversaries* $\mathcal{A}$, *there exists a polynomial-size simulator* $\mathcal{S}$ *such that the following distributions are computationally indistinguishable:*

$$\left\{\left\langle \widetilde{\mathrm{C}}, \widetilde{\mathbf{x}}, \mathbf{dk}\right\rangle : (\widetilde{\mathrm{C}}, \mathbf{dk}, \mathrm{sk}) \leftarrow \mathsf{Grb}\big(1^k, \mathrm{C}\big); \mathbf{x} \leftarrow \mathcal{A}(1^k); \widetilde{\mathbf{x}} \leftarrow \mathsf{Enc}(\mathrm{sk}, \mathbf{x})\right\},$$

$$\left\{\left\langle \widetilde{\mathrm{C}}, \widetilde{\mathbf{x}}, \mathbf{dk}\right\rangle : \mathbf{x} \leftarrow \mathcal{A}(1^k); (\widetilde{\mathrm{C}}, \widetilde{\mathbf{x}}, \mathbf{dk}) \leftarrow \mathcal{S}\big(\mathrm{C}, f(\mathbf{x})\big)\right\}.$$

Adaptive input privacy. While non-adaptive privacy is sufficient for some applications (e.g., secure two-party computation in the semi-honest model) there are other useful applications for which it falls short. This typically occurs in situations where the adversary can choose its inputs *as a function of the garbled circuit* (for example in one-time programs [11]). The following simulation-based definition of adaptive input privacy guarantees that the garbled circuit, the encoded input and the decoding keys are all simulatable given only the circuit and the result of the computation. Like the non-adaptive definition, this holds for adversarially-chosen inputs; but, unlike the non-adaptive definition, the inputs can be chosen as a function of the garbled circuit.

Definition 3 (Sim2-security). *A garbling scheme* $\mathsf{Garb} = (\mathsf{Grb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* Sim2*-secure with respect to a circuit* C *if, for all polynomial-size adversaries* $\mathcal{A}$, *there exists a polynomial-size stateful simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that the following distributions are computationally indistinguishable:*

$$\left\{\left\langle \widetilde{\mathrm{C}}, \widetilde{\mathbf{x}}, \mathbf{dk}, st_{\mathcal{A}}\right\rangle : (\widetilde{\mathrm{C}}, \mathbf{dk}, \mathrm{sk}) \leftarrow \mathsf{Grb}\big(1^k, \mathrm{C}\big); st_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathrm{sk}, \cdot)}(\widetilde{\mathrm{C}}, \mathbf{dk})\right\},$$

$$\left\{\left\langle \widetilde{\mathrm{C}}, \widetilde{\mathbf{x}}, \mathbf{dk}, st_{\mathcal{A}} \right\rangle : (\widetilde{\mathrm{C}}, \mathbf{dk}) \leftarrow \mathcal{S}_1(\mathrm{C}); st_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathrm{OSIM}^{\mathcal{S}_2}(\cdot)}(\widetilde{\mathrm{C}}, \mathbf{dk})\right\},$$

where $\mathrm{OSIM}^{\mathcal{S}_2}$ *is a single-query oracle that takes as input* $\mathbf{x}$ *and returns* $\widetilde{\mathbf{x}} \leftarrow \mathcal{S}_2(\mathrm{C}, f(\mathbf{x}))$.

5 Garbling Schemes via Structured Encryption

The first step in our framework is to describe the functionality f as a *structured circuit* which, roughly speaking, is a circuit with gates that can query data structures that support a given set of operations. Given a structured circuit representation of f we then garble it using an appropriate set of structured encryption schemes.

Structured circuits. An n input and m output structured circuit C over a basis $\mathcal{B} = \{\mathscr{T}_1, \ldots, \mathscr{T}_\beta\}$ is a directed acyclic graph with n input wires and m output wires such that each gate g has access to a data structure of type $\mathscr{T} \in \mathcal{B}$ which supports an operation $\mathsf{Query} : \mathcal{U} \times \mathcal{Q}_1 \times \cdots \times \mathcal{Q}_\nu \to \mathcal{R}$. We say that g is a $(\mathscr{T}, \nu)$-gate if: (1) it has access to a structure δ of type $\mathscr{T}$; and (2) it has ν input wires that carry queries $(q_1, \ldots, q_\nu) \in \mathcal{Q}_1 \times \cdots \times \mathcal{Q}_\nu$ and an output wire that carries answers in $\mathcal{R}$. We require that if g_1's output wire is g_2's ith input wire, then $\mathcal{R}_1 = \mathcal{Q}_{2,i}$ where $\mathcal{R}_1$ refers to the response space of g_1 and $\mathcal{Q}_{2,i}$ denotes the ith query space of g_2.

Throughout, we assume a topological ordering on C and denote its ith gate by g_i. For notational convenience, we sometimes write $\mathscr{T}(g)$, $\mathcal{Q}(g)$, $\mathcal{R}(g)$ to refer to a gate g's type, query space and answer space, respectively.

A structured circuit C is evaluated on input $(q_1, \ldots, q_n)$ from the input wires to the output wires. When the inputs to the incoming wires of a gate g have been obtained, the output wire of g is set to $a := \mathsf{Query}(\delta, q_1, \ldots, q_\nu)$. The output of the circuit are the values obtained on the output wires of the circuit.

5.1 Our Framework

We now describe our approach to designing special-purpose garbled circuits. Let C be a structured circuit over the basis $\mathcal{B}$ and let $\{\mathsf{SE}_1, \ldots, \mathsf{SE}_\beta\}$ be a set of structured encryption schemes for each abstract data type in $\mathcal{B}$. Our approach is described in detail in Fig. 1 and, at a high level, works as follows.

If g has access to a data structure δ of type $\mathscr{T}$ (e.g., a graph or a matrix) then δ is encrypted using a structured encryption scheme for type $\mathscr{T}$. The resulting encrypted structure γ is the garbled gate and the tokens for queries q are used as encodings for the wires. To allow for the connection of gates to one another, the underlying structured encryption schemes must be associative.

Intuitively, the input privacy of the resulting garbled (structured) circuit is guaranteed by the security of the structured encryption scheme. This approach results in garbled circuits that have the same size as the *structured* circuit for f. For certain functions, the structured circuit representation can be much smaller than the boolean circuit representation (we discuss this further in Section 6).

Let $\mathcal{B} = \{\mathscr{T}_1, \ldots, \mathscr{T}_\beta\}$ be a basis and $(\mathsf{SE}_1, \ldots, \mathsf{SE}_\beta)$ be associative structured encryption schemes for the types in $\mathcal{B}$. Construct a garbling scheme $\mathsf{Garb} = (\mathsf{Grb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ for the class of n input and m output structured circuits over $\mathcal{B}$ as follows:

- $\mathsf{Grb}(1^k, \mathrm{C})$:
  1. **(output gates)** let $\mathrm{OUT} = (o_1, \ldots, o_m)$ be the set of output gates and for each o_i,
     (a) generate a key $K_i \leftarrow \mathsf{SE}_{\mathscr{T}(o_i)}.\mathsf{Gen}(1^k)$
     (b) for all $a \in \mathcal{R}(o_i)$, sample $\lambda_{i,a} \xleftarrow{\$} \{0,1\}^k$
     (c) compute $\gamma_i \leftarrow \mathsf{SE}_{\mathscr{T}(o_i)}.\mathsf{Enc}_{K_i}(\delta, \boldsymbol{\lambda})$, where δ is the structure of o_i and $\boldsymbol{\lambda} = \{\lambda_{i,a}\}_{a \in \mathcal{R}(o_i)}$
     (d) set dk_i to be a lookup table that maps $\lambda_{i,a}$ to a.
  2. **(non-output gates)** let $\overline{\mathrm{OUT}} = (g_1, \ldots, g_\ell)$ be the set of non-output gates and for each g_i,
     (a) generate a key $K_i \leftarrow \mathsf{SE}_{\mathscr{T}(g_i)}.\mathsf{Gen}(1^k)$
     (b) let d be the descendant of g_i and let K_d be the key generated for it
     (c) for all $q \in \mathcal{Q}(d)$, compute $\tau_q := \mathsf{SE}_{\mathscr{T}(d)}.\mathsf{Token}_{K_d}(q)$
     (d) compute $\gamma_i \leftarrow \mathsf{SE}_{\mathscr{T}(g_i)}.\mathsf{Enc}_{K_i}(\delta, \boldsymbol{\tau})$, where $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{|\mathcal{Q}(d)|})$.
  3. let $\widetilde{\mathrm{C}} = (\gamma_1, \ldots, \gamma_{|\mathrm{C}|})$, where γ_j is the garbling of the jth gate in C
  4. if $\mathrm{IN} = (g_1^\star, \ldots, g_n^\star)$ are the inputs gates, let $\mathrm{sk} = (K_1^\star, \ldots, K_n^\star)$ be the keys generated for these gates
  5. let $\mathbf{dk} = (\mathrm{dk}_1, \ldots, \mathrm{dk}_m)$,
  6. output $(\widetilde{\mathrm{C}}, \mathbf{dk}, \mathrm{sk})$
- $\mathsf{Enc}(\mathrm{sk}, x)$: compute $\tau \leftarrow \mathsf{SE}_{\mathscr{T}(x)}.\mathsf{Token}_{K(x)}(x)$ and output $\widetilde{x} = \tau$.
- $\mathsf{Eval}(\widetilde{\mathrm{C}}, \widetilde{\mathbf{x}})$: evaluate $\widetilde{\mathrm{C}}$ from the input wires to the output wires as follows: when the tokens τ_1 and τ_2 of the incoming wires to a garbled gate γ have been obtained, set the output wire of γ to $\tau_3 \leftarrow \mathsf{SE}_{\mathscr{T}(\gamma)}.\mathsf{Query}_{\mathsf{e}}(\gamma, \tau_1, \tau_2)$. After processing all gates, output $\widetilde{\mathbf{y}} = (\lambda_{o_1}, \ldots, \lambda_{o_m})$, where λ_{o_i} is the value obtained on the output wire of the ith output gate o_i.
- $\mathsf{Dec}(\widetilde{\mathbf{x}}, \mathrm{dk}_i, \widetilde{y}_i)$: parse $\widetilde{y}_i$ as λ_i and output $y_i := \mathrm{dk}_i[\lambda_i]$.

Fig. 1. A framework for designing special-purpose garbled circuits.

Non-adaptive input privacy. We show that if the underlying structured encryption schemes are CQA1-secure, then our construction results in a garbling scheme that provides non-adaptive input privacy. Due to space restrictions, the proof is deferred to the full version of this work.

Theorem 1. *If* $(\mathsf{SE}_1, \ldots, \mathsf{SE}_\beta)$ *are* CQA1*-secure, then the scheme described in Fig.1 is* SIM1*-secure.*

Adaptive input privacy. In the full version of this work, we also show that if the underlying schemes are CQA2-secure, then the resulting garbling scheme provides the stronger notion of adaptive input privacy.

Theorem 2. *If* $(\mathsf{SE}_1, \ldots, \mathsf{SE}_\beta)$ *are* CQA2*-secure, then the construction described in Fig.1 is* SIM2*-secure.*

A remark on Yao's construction. We observe that Yao's garbled circuit construction can be viewed as an instantiation of our framework using 2×2 matrix encryption schemes that support lookup queries. Recall that in Yao's construction, garbled circuits are constructed as follows. Each gate g in the circuit C is replaced with a garbled gate $\widetilde{g}$. Here, we assume without loss of generality that g has two input wires w_a and w_b and one output wire w_c. The bit values conducted by each wire are replaced with a randomly chosen encoding. So the 0 and 1 bits on wire w_a are encoded as ω_0^a and ω_1^a which are sampled uniformly at random. The encodings for all the bits of w_b and w_c are generated similarly. The garbled gate $\widetilde{g}$ is constructed such that, given (ω_0^a, ω_0^b) it returns $\omega_{g(0,0)}^c$, given (ω_0^a, ω_1^b) it returns $\omega_{g(0,1)}^c$, and so on. Notice that because the encodings are chosen uniformly at random, they do not reveal any information about the real wire values.

These garbled gates can be viewed as structured encryption schemes for 2×2 matrices that support lookups. To illustrate this, we briefly sketch how each implies the other (we defer a more formal treatment to the full version of this work). Given a Boolean gate g we can construct a garbled gate $\widetilde{g}$ using any associative 2-dimensional matrix encryption scheme for 2×2 matrices. The 0 and 1 labels for w_a are tokens for the first and second row, respectively; and the 0 and 1 labels for w_b are tokens for the first and second column, respectively. The garbled gate $\widetilde{g}$ is then the encryption of the matrix M defined as $M[i,j] = \tau_{g(i-1,j-1)}^c$, where τ_0^c and τ_1^c are the tokens used as encodings for g's output wire (alternatively, for one of its descendent's input wires). In the other direction, we can construct an associative 2×2 matrix encryption scheme from any garbled gate construction. It suffices to view the garbled gate as the encrypted matrix (replacing the output wire encodings with the associated data) and the input wire encodings as the tokens for lookup queries.

6 Concrete Constructions

In the previous Section, we showed how to construct special-purpose garbled circuits for any function f that can be written as a structured circuit over a

basis $\mathcal{B}$. This requires, however, that we have structured encryption schemes for the data types in $\mathcal{B}$. In [8], several structured encryption schemes were proposed including a matrix encryption scheme that supports lookup queries, a graph encryption scheme that supports adjacency queries (i.e., given two nodes, test whether they are adjacent), a graph encryption scheme that supports neighbor queries (i.e., given a node, return all of its neighbors) and a web-graph encryption scheme that supports focused subgraph queries. All the schemes in [8] were shown CQA2-secure so, using our framework, we get adaptively-secure special-purpose garbling schemes for any structured circuit over the basis $\mathcal{B}$ consisting of the data types mentioned above.

This leads to garbling schemes and (when combined with OT in the natural way) special-purpose two-party protocols in the semi-honest model for several graph-based functionalities. Note that in all these functionalities there is a set of *public* vertices V, and one player holds a private set of edges E over V that the second player wants to query in some way. This captures several real-life scenarios, e.g., in online social network analysis where the identities of users is public (e.g., Facebook, LinkedIn, Google+) but the relationships between users (i.e., friendships, connections, relationships) is private. In particular, this leads to two-party protocols for the following functionalities:

- (neighbor queries) $f_V(E, v) = (\perp, \Gamma(v))$, where $\Gamma(v)$ are the neighbor of v.
- (adjacency queries) $f_V(E, (v_1, v_2)) = (\perp, M[v_1, v_2])$, where M_G is the adjacency matrix of $G = (V, E)$.
- (focused subgraph queries) $f_V((E, D_1, \ldots, D_{|V|}), w) = (\perp, \Sigma(w))$, where D_1 through $D_{|V|}$ are documents (e.g., user profiles) associated with the vertices in V and $\Sigma(w) = \{v_i \in V : w \in D_i\} \cup \{\Gamma(v_i) \subseteq V : w \in D_i\}$, i.e., the vertices whose documents contain the keyword w and their neighbors.

We briefly note that in the context of online social networks, focused subgraph queries (FSQ) allow P_2 to make queries of the type "search for all users who are friends with someone that likes product X", which is particularly compelling for marketing applications. In the context of healthcare (i.e., the vertices are patients and the documents are their medical records), FSQs allow P_2 to query for all patients who are related to someone who has a particular disease or symptom.

In addition to the schemes mentioned above, we can use our framework to design special-purpose garbling schemes (and therefore two-party protocols) for functionalities not handled by the structured encryption schemes of [8]. This includes Boolean circuits, DFAs and BPs. In fact, in the full version, we show that all these functionalities can be handled using a 2-dimensional matrix encryption scheme. Due to space restrictions, we only describe this new matrix encryption construction and leave its application to Boolean circuits, DFAs and BPs—which is straightforward—to the full version of this work.

While [8] show how to construct an associative matrix encryption scheme that is CQA2-secure, their particular construction is not appropriate for our purpose. More precisely, their scheme is only one-dimensional, in the sense that it only generates a single token for a lookup query (i, j). On the other, for our

purposes, we need a scheme that generates independent tokens for i and j that can later be combined to do a lookup at location (i, j) on the encrypted matrix.

1-D Matrix encryption. At a high level, the scheme from [8] works as follows: given an $n \times m$ matrix M a new matrix C is constructed such that each element $M[i, j]$ is stored in C at location $(\alpha, \beta) := P_{K_1}(i, j)$ encrypted under key $K_{\alpha,\beta} := F_{K_2}(\alpha, \beta)$, where $P : \{0,1\}^k \times [n] \times [m] \to [n] \times [m]$ is a pseudo-random permutation [2] $F : \{0,1\}^k \times [n] \times [m] \to \{0,1\}^{\ell(k)}$ is a pseudo-random function. The encrypted matrix is C and a lookup token for location (i, j) consists of the tuple $(\alpha, \beta, F_{K_2}(\alpha, \beta))$.

2-D Matrix encryption. We sketch here how to make the scheme from [8] two-dimensional. Note that this approach only yields a CQA1-secure scheme. This, however, implies a SIM1-secure garbling scheme which is sufficient for important applications like two-party computation.

Let $\ell(k)$ be an upper bound on the length of the information stored in the matrix (e.g., the associated data). We use a primitive introduced by Naor and Reingold in [17] called a pseudo-random synthesizer, which can be built from weak pseudo-random functions. A synthesizer Synth is an efficiently computable function such that

$$\left\{ \left\langle \mathsf{Synth}(x_i, y_j) \right\rangle_{1 \le i,j \le m} : \mathbf{x} \xleftarrow{\$} X^n; \mathbf{y} \xleftarrow{\$} X^n \right\} \overset{c}{\approx} \left\{ \langle \mathbf{r} \rangle : \mathbf{r} \xleftarrow{\$} X^{n^2} \right\}.$$

Let P and Q be two pseudo-random permuations and let F be a pseudo-random function. In the new scheme the element $M[i, j]$ is stored at location $(\alpha, \beta) := (P_{K_1}(i), Q_{K_2}(j))$ in C and XORed with the pad $K_{i,j} := \mathsf{Synth}(F_{K_3}(0\|\alpha), F_{K_3}(1\|\beta))$. Lookup tokens for location (i, j) are simply $(\alpha, F_{K_3}(0\|\alpha))$ and $(\beta, F_{K_3}(1\|\beta))$. It is easy to show that this scheme is CQA1-secure (we defer a proof to the full version) so by, Theorem 1, it can be used to construct a SIM1-secure garbling schemes. If SIM2-security is needed one can use the transformation of [3].

References

1. B. Applebaum, Y. Ishai, and E. Kushilevitz. How to garble arithmetic circuits. In *Symposium on Foundations of Computer Science (FOCS '11)*, pages 120–129. IEEE Computer Society, 2011.
2. M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. Secure evaluation of private linear branching programs with medical applications. In *European Symposium on Research in Computer Security (ESORICS '09)*, pages 424–439, 2009.
3. M. Bellare, V. T. Hoang, and P. Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *Advances in Cryptology - ASIACRYPT '12*, 2012.

[2] Note that pseudo-random permutations over small domains can be constructed using techniques from [5].

4. M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security (CCS '12)*, pages 784–796, 2012.
5. J. Black and P. Rogaway. Ciphers with arbitrary finite domains. In B. Preneel, editor, *The Cryptographers' Track at the RSA Conference (CT-RSA '02)*, volume 2271 of *Lecture Notes in Computer Science*, pages 114–130. Springer-Verlag, 2002.
6. J. Brickell, D. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *ACM Conference on Computer and Communications Security (CCS '07)*, pages 498–507. ACM, 2007.
7. Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security (ACNS '05)*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455. Springer, 2005.
8. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.
9. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. ACM, 2006.
10. E-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See http://eprint.iacr.org/2003/216.
11. S. Goldwasser, Y. Kalai, and G. Rothblum. One-time programs. In *Advances in Cryptology - CRYPTO 2008*, pages 39–56. Springer-Verlag, 2008.
12. Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *IEEE Symposium on Foundations of Computer Science (FOCS '00)*, pages 294–304. IEEE Press, 2000.
13. Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *International Colloquium on Automata, Languages and Programming (ICALP '02)*, pages 244–256. Springer, 2002.
14. L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *ACM Conference on Computer and Communications Security (CCS '06)*, pages 410–420. ACM, 2006.
15. P. Mohassel, S. Niksefat, S. Sadeghian, and B. Sadeghiyan. An efficient protocol for oblivious dfa evaluation and applications. In *RSA Conference - Cryptographer's Track (CT-RSA '12)*, 2012.
16. M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *Symposium on Theory of Computing (STOC '01)*, pages 590–599. ACM, 2001.
17. M. Naor and O. Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *Symposium on Foundations of Computer Science (FOCS '95)*, pages 170–. IEEE Computer Society, 1995.
18. D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.
19. A. Yao. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science (FOCS '86)*, pages 162–167. IEEE Computer Society, 1986.