Valiant's Universal Circuit is Practical

Ágnes Kiss(✉) and Thomas Schneider

TU Darmstadt, Darmstadt, Germany
{agnes.kiss,thomas.schneider}@crisp-da.de

Abstract. Universal circuits (UCs) can be programmed to evaluate any circuit of a given size k. They provide elegant solutions in various application scenarios, e.g. for private function evaluation (PFE) and for improving the flexibility of attribute-based encryption (ABE) schemes. The optimal size of a universal circuit is proven to be $\Omega(k \log k)$. Valiant (STOC'76) proposed a size-optimized UC construction, which has not been put in practice ever since. The only implementation of universal circuits was provided by Kolesnikov and Schneider (FC'08), with size $\mathcal{O}(k \log^2 k)$.

In this paper, we refine the size of Valiant's UC and further improve the construction by (at least) $2k$. We show that due to recent optimizations and our improvements, it is the best solution to apply in the case for circuits with a constant number of inputs and outputs. When the number of inputs or outputs is linear in the number of gates, we propose a more efficient hybrid solution based on the two existing constructions. We validate the practicality of Valiant's UC, by giving an example implementation for PFE using these size-optimized UCs.

Keywords: Universal circuit · Size-optimization · Private function evaluation

1 Introduction

Any computable function $f(x)$ can be represented as a Boolean circuit with input bits $x = (x_1, \ldots, x_u)$. Universal circuits (UCs) are programmable circuits, which means that beyond the true u inputs, they receive $p = (p_1, \ldots, p_m)$ program bits as further inputs. By means of these program bits, the universal circuit is programmed to evaluate the function, such that $UC(x, p) = f(x)$. The advantage of universal circuits in general is that one can apply the same UC for computing different functions of the same size. An analogy between universal circuits and a universal Turing machine allows to turn any function into data in the form of a program description. Thus, the size-depth problem of UCs can be related to the time-space problem for Turing machines [Val76].

Efficient constructions considering both the size and the depth of the UC were proposed. The first approach was the optimization of the size by Valiant [Val76], resulting in a construction with asymptotically optimal size $\mathcal{O}(k \log k)$ and depth $\mathcal{O}(k)$, where k denotes the size of the simulated circuits. The second optimization

M. Fischlin and J.-S. Coron (Eds.): EUROCRYPT 2016, Part I, LNCS 9665, pp. 699–728, 2016.
DOI: 10.1007/978-3-662-49890-3_27

was proposed with respect to the UC depth in [CH85], where a construction with linear depth $\mathcal{O}(d)$ in the simulated circuit depth d and size $\mathcal{O}(\frac{k^3 d}{\log k})$ was designed. In this paper, due to the applications that we revisit in Sect. 1.2, e.g., diagnostic programs, blinded policies and database queries, we concentrate on the existing size-optimized UCs and note, that the asymptotically optimal size is $\Omega(k \log k)$ [Val76, Weg87].

The most prominent application of universal circuits is the evaluation of private functions based on *secure function evaluation* (SFE) or *secure two-party computation*. SFE enables two parties P_1 and P_2 to evaluate a publicly known function $f(x, y)$ on their private inputs x and y, ensuring that none of the participants learns anything about the other participant's input. SFE ensures that both P_1 and P_2 learn the correct result of the evaluation. Many secure computation protocols use Boolean circuits for representing the desired functionality, such as Yao's garbled circuit protocol [Yao86, LP09a] and the GMW protocol [GMW87]. In some applications the function itself should be kept secret. This setting is called *private function evaluation* (PFE), where we assume that only one of the parties P_1 knows the function $f(x)$, whereas the other party P_2 provides the input to the private function. P_2 learns no information about f besides the size of the circuit defining the function and the number of inputs and outputs.

PFE can be reduced to SFE [AF90, SYY99, Pin02, KS08b] by securely evaluating a UC that is programmed by P_1 to evaluate the function f on P_2's input x. Thus, P_1 provides the program bits for the UC and P_2 provides his private input x into an SFE protocol that computes a UC. The complexity of PFE in this case is determined mainly by the complexity of the UC construction. The security follows from that of the SFE protocol that is used to evaluate the UC. If the SFE protocol is secure against semi-honest, covert or malicious adversaries, then the PFE protocol is secure in the same adversarial setting.

1.1 Related Work on Universal Circuits and Private Function Evaluation

Universal Circuits. Valiant presented an asymptotically optimal universal circuit construction with size $\approx 4.75(u+v+k^*)\log_2(u+v+k^*)$ [Val76], relying on edge-universal graphs. u, k and v denote the respective number of inputs, gates and outputs in the simulated circuit, and k^* is the number of gates in the equivalent fanout-2 circuit, with $k \leq k^* \leq 2k+v$. Valiant's size-optimized UC construction was recapitulated in [Weg87, Sect. 4.8]. However, Valiant's construction has been considered to be mostly a proof of existence of a universal circuit, whereas details needed for the practical realization, e.g., how to derive the program for the UC are left open. Kolesnikov and Schneider proposed a UC construction with size $\approx 0.75k\log_2^2 k + 2.25k\log_2 k + k\log_2 u + (0.5k+0.5v)\log_2 v$ [KS08b, Sch08]. They present the first implementation of PFE using UCs by extending the Fairplay secure computation framework [MNPS04]. Some building blocks of this construction are of interest, but due to its asymptotically non-optimal size, we show in Sect. 3.2 that Valiant's UC construction results in smaller UCs for circuits in

the most general case. The UC constructions from [Val76, KS08b] were generalized for circuits consisting of gates with more than two inputs in [SS08]. In this paper, we show the practicality of Valiant's UC construction.

In concurrent and independent work [LMS16], Lipmaa et al. also bring the same UC construction to practice. They detail a k-way recursive construction for UCs, instantiate it for $k \in \{2, 4\}$ as in [Val76], and descrease its *total* number of gates compared to that of Valiant's construction. However, in contrast to our optimizations, their number of AND gates is exactly the same and therefore their improvement does not affect PFE with UC, when XOR gates are evaluated for free [KS08a]. Currently their implementation for generating and programming UCs supports the 2-way recursive construction, the same construction that we study and realize in practice in this work.

Private Function Evaluation. In [KM11], Katz and Malka presented an approach for PFE that does not rely on UCs. They use (singly) homomorphic public-key encryption as well as a symmetric-key encryption scheme and achieve constant-round PFE with linear communication complexity. However, the number of public-key operations is linear in the circuit size and due to the gap between the efficiency of public-key and symmetric-key operations, this results in a less efficient protocol for circuits with reasonable size. Their protocol is secure against semi-honest adversaries and uses Yao's garbled circuit technique [Yao86]. Mohassel and Sadeghian consider PFE with semi-honest adversaries in [MS13]. Their generic PFE framework can be instantiated with different secure computation protocols. The first version uses homomorphic encryption with which they achieve linear complexity in the circuit size and the second alternative relies solely on oblivious transfers (OT), that results in a method with $\mathcal{O}(k \log k)$ symmetric-key operations, where k denotes the circuit size. The OT-based construction is more desirable in practice, since using OT extension, the number of expensive public-key operations can significantly be reduced, s.t. it is independent of the number of OTs [IKNP03, ALSZ13]. The asymptotical complexity of the OT-based construction of [MS13] and Valiant's UCs for PFE is the same, and therefore we compare these solutions for PFE in more detail in Sect. 4.2. Mohassel et al. extend the framework from [MS13] to malicious adversaries in [MSS14] and show that an actively secure PFE framework with linear complexity $\mathcal{O}(k)$ is feasible, using singly homomorphic encryption.

1.2 Applications of Universal Circuits

Universal circuits have several applications, which we summarize in this section.

Private Function Evaluation. As mentioned before, UCs can be used to securely evaluate a private function using a generic secure computation protocol. [CCKM00] shows an application for secure computation, where evaluating UCs or other PFE protocols would ensure privacy: when *autonomous mobile agents* migrate between several distrusting hosts, the privacy of the inputs of the hosts is achieved using SFE, while privacy of the mobile agent's code can be

guaranteed with PFE. Privacy-preserving *credit checking* using garbled circuits is described in [FAZ05]. Their original scheme cannot represent any policy, though by evaluating a UC, their scheme can be extended to more complicated credit checking policies. [OI05] show a method to *filter remote streaming data* obliviously, using secret keywords and their combinations. Their scheme can additionally preserve data privacy by using PFE to search the matching data with a private search function. Privacy-preserving evaluation of *diagnostic programs* was considered in [BPSW07], where the owner of the program does not want to reveal the diagnostic method and the user does not want to reveal his data. Example applications for such programs include medical systems [BFK+09] and remote software fault diagnosis, where in both cases the function and the user's input are desired to be handled privately. In the protocol presented in [BPSW07], the diagnostic programs are represented as binary decision trees or branching programs which can easily be converted into a Boolean circuit representation and evaluated using PFE based on universal circuits. Besides, PFE can be applied to create *blinded policy evaluation protocols* [FAL06,FLA06]. [FAL06] utilizes UCs for so-called oblivious circuit policies and [DDKZ13] for hiding the circuit topology in order to create one-time programs. Since PFE using UCs utilizes general secure computation protocols, it is possible to outsource the function and the data to two or multiple servers (using XOR secret sharing) and then run private queries on these. This is not directly possible with other PFE protocols, e.g., with the protocol presented in [KM11] or the homomorphic encryption-based protocols from [MS13,MSS14].

Beyond Private Function Evaluation. Besides being used for PFE, UCs can be applied in various other scenarios. Efficient *verifiabile computation* on encrypted data was studied in [FGP14]. A verifiable computation scheme was proposed for arbitrary computations and a UC is required to hide the function. [GGPR13] make use of universal circuits for reducing the verifier's preprocessing step. In [GHV10], a *multi-hop homomorphic encryption* scheme is proposed that also uses a universal circuit evaluator to achieve the privacy of the function. When the common reference string is dependent on a function that the verifier is interested in outsourcing, then the function description can be provided as input to a UC of appropriate size. In [PKV+14,FVK+15], universal circuits are used for hiding *queries in database management systems* (DBMSs). The Blind Seer DBMS was improved in [PKV+14] by making use of a simpler UC for evaluating queries, which does not hide the circuit topology. The authors mention that in case the topology of the SQL formula and the circuit have to be kept private, a UC can be utilized. As described in [Att14], the *Attribute-Based Encryption* (ABE) schemes for some polynomial-size circuits can be turned into ciphertext-policy ABE by using universal circuits. The ABE scheme of [GGHZ14] also uses UCs.

1.3 Outline and Our Contributions

In Sect. 2, we revisit the two existing size-optimized UC constructions of [Val76,KS08b]. We put an emphasis on the asymptotically size-optimal

method proposed by Valiant in [Val76]. This complex construction makes use of an internal graph representation and programs a so-called edge-universal graph. However, the algorithm for programming a universal circuit is not explicitly described and in the presence of the included optimizations is not straightforwardly applicable. In Sect. 2.1, we recapitulate Valiant's recursive edge-universal graph construction and describe how the construction of UCs can be reduced to this problem. In Sect. 2.2, we briefly summarize the main building blocks of the UC construction of [KS08b].

Optimized Size and Depth of Valiant's UC Construction: In Sect. 3, we elaborate on the concrete size of Valiant's UC construction. We refine upper and lower bounds for the size of the edge-universal graph and approximate a closed formula with $\leq 2\,\%$ deviation from the actual size in Sect. 3.1. We include two optimizations detailed in Sect. 3.2, achieving altogether a linear improvement of at least $4u+4v+2k$. We give hybrid constructions for cases with many inputs and outputs in the same section. In Sect. 3.2, we compare the refined concrete size and the depth of Valiant's construction with that of [KS08b] and conclude the advantage of Valiant's method (potentially using building blocks from [KS08b]).

Valiant's Size-Optimized UC Construction in Practice: In Sect. 4, we detail the steps of our algorithm for a practical realization of Valiant's UC construction and provide an example application for PFE. We describe the internal representations and the algorithms in our UC compiler in Sect. 4.1, along with detailed implementations of universal gates and switches. We compare our resulting PFE with the OT-based protocol from [MS13] in Sect. 4.2. We show concrete example circuits and elaborate on the number of symmetric-key operations and the performance of our UC compiler.

2 Existing Universal Circuit Constructions

In this section, we summarize the two size-optimized universal circuit constructions: of [Val76] in Sect. 2.1 and of [KS08b] in Sect. 2.2.

2.1 Valiant's Universal Circuit Construction

In this section, we describe Valiant's edge-universal graph construction for graphs for which all nodes have at most one incoming and at most one outgoing edge and detail how two such graphs can be used for constructing universal circuits [Val76].

Edge-Universal Graphs. $G = (V, E)$ is a directed graph with the set of nodes $V = \{1, \ldots, n\}$ and the set of edges $E \subseteq V \times V$. A directed graph has *fanin* or *fanout* ℓ if each of its nodes has at most ℓ edges directed into or out of it, respectively. $\Gamma_\ell(n)$ denotes the set of all acyclic directed graphs with n nodes and fanin and fanout ℓ. Further on, we require a labelling of the nodes in a

topological order, i.e., $i > j$ implies that there is no directed path from i to j. In a graph in $\Gamma_\ell(n)$, a topological ordering can always be found with computational complexity $\mathcal{O}(n + \ell n)$.

An *edge-embedding* of graph $G = (V, E)$ into $G' = (V', E')$ is a mapping that maps V into V' one-to-one, with possible additional nodes in V', and E into directed paths in E', such that they are pairwise edge-disjoint, i.e., an edge can be used only in one path. A graph G' is *edge-universal* for $\Gamma_\ell(n)$ if it has distinguished poles $\{p_1, \ldots, p_n\} \subseteq V'$ and every graph $G \in \Gamma_\ell(n)$ with node set $V = \{1, \ldots, n\}$ can be edge-embedded into G' by a mapping φ^G such that $\varphi^G : i \mapsto p_i$ and $\varphi^G : (i, j) \mapsto \{\text{path from pole } p_i \text{ to pole } p_j\}$ for each $i, j \in V$.

Here, we recapitulate Valiant's construction for acyclic edge-universal graph for $\Gamma_1(n)$, denoted by U_n, that has fewer than $2.5n \log_2 n$ nodes, fanin and fanout 2 and poles with fanin and fanout 1. Valiant presents another edge-universal graph construction with a lower multiplicative constant $2.375n \log_2 n$. We omit that version of the algorithm for two reasons: firstly, our aim is to show the practicality of Valiant's approach and secondly, including all the optimizations even in the simpler construction is a challenging task in practice. The more efficient algorithm uses four subgraphs instead of two at each recursion and utilizes a skeleton with a more complex structure. For more details on this improved algorithm, the reader is referred to [Val76,LMS16]. We leave showing the practicality of the improved method as future work.

Valiant's Edge-Universal Graph Construction for $\Gamma_1(n)$ Graphs: The edge-universal graph for $\Gamma_1(n)$, denoted by U_n, is constructed with poles $\{p_1, \ldots, p_n\}$ with fanin and fanout 1, which are connected according to the skeleton shown in Figs. 1a and b. The poles are emphasized as special nodes with squares, and the additional nodes are shown as circles. The recursive construction works as follows: the nodes denoted by $\{q_1, \ldots, q_{\lceil \frac{n-2}{2} \rceil}\}$ and $\{r_1, \ldots, r_{\lfloor \frac{n-2}{2} \rfloor}\}$ are considered as the poles of two smaller edge-universal graphs called subgraphs $Q_{\lceil \frac{n-2}{2} \rceil}$ and $R_{\lfloor \frac{n-2}{2} \rfloor}$, respectively, that are otherwise not shown. Since they are poles of the two subgraphs with such a skeleton but not of U_n, they will have at most the allowed fanin and fanout 2: they inherit one incoming and one outgoing edge from the outer skeleton, and at most one incoming and one outgoing edge from the subgraph. $Q_{\lceil \frac{n-2}{2} \rceil}$ (and $R_{\lfloor \frac{n-2}{2} \rfloor}$) is then constructed similarly: the skeleton is completed and two smaller graphs with sizes $\lceil \frac{\lceil \frac{n-2}{2} \rceil - 2}{2} \rceil$ and $\lfloor \frac{\lceil \frac{n-2}{2} \rceil - 2}{2} \rfloor$ (and sizes $\lceil \frac{\lfloor \frac{n-2}{2} \rfloor - 2}{2} \rceil$ and $\lfloor \frac{\lfloor \frac{n-2}{2} \rfloor - 2}{2} \rfloor$) are constructed. For starting off the recursion, U_1 is a graph with a single pole while U_2 and U_3 are graphs with two and three connected poles, respectively. Valiant gives special constructions for U_4, U_5 and U_6 and shows that it is possible to obtain the respective edge-universal graphs with altogether 3, 7 and 9 additional nodes, respectively, as shown in Figs. 1c, d, and e.

We recapitulate the proof from [Val76] that U_n is edge-universal for $\Gamma_1(n)$, such that any graph with n nodes and fanin and fanout 1 can be edge-embedded into U_n. According to the definition of edge-embedding, it has to be shown that given any $\Gamma_1(n)$ graph G with set of edges E, for any $(i, j) \in E$ and $(k, l) \in E$

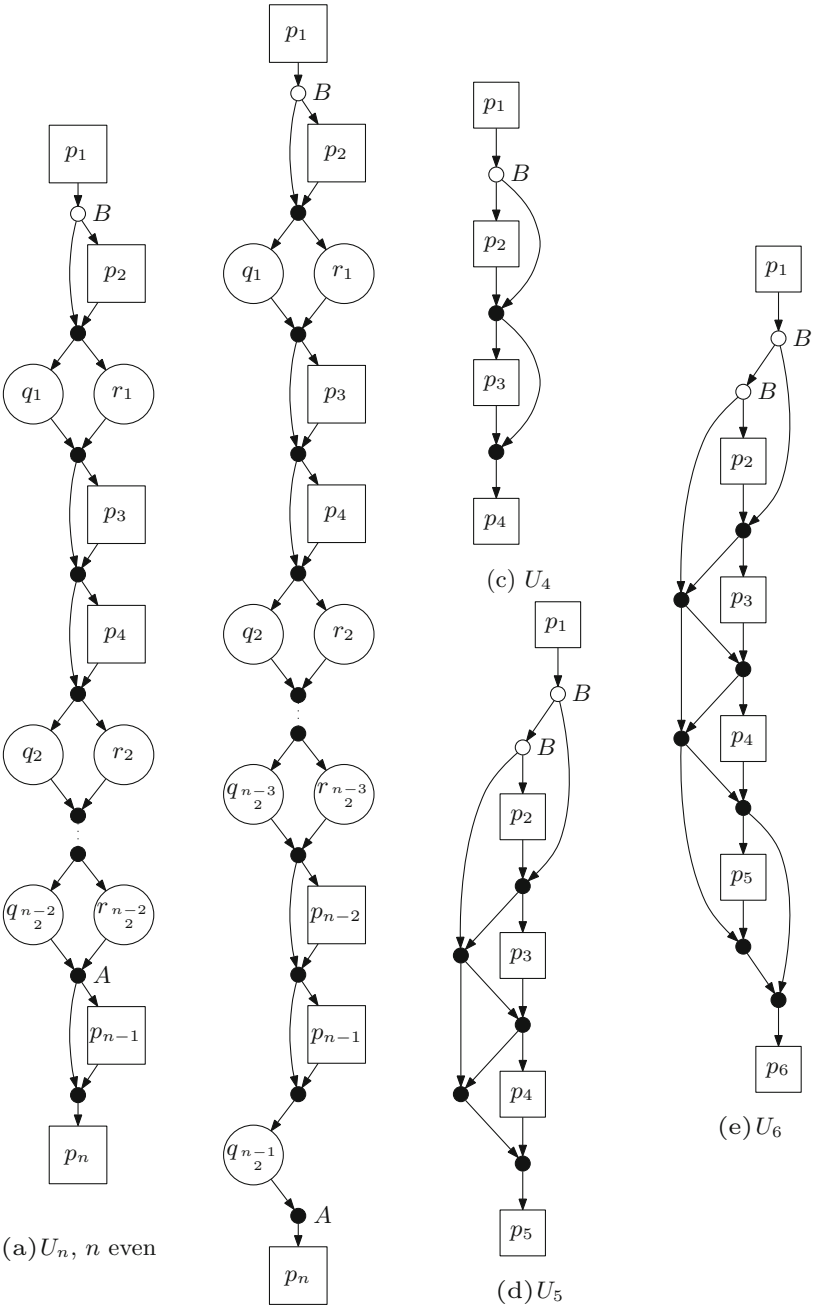

Fig. 1. Skeleton of Valiant's edge-universal graph and optimized cases.

we can find pairwise edge-disjoint paths from p_i to p_j and from p_k to p_l in U_n. As before, the labelling of nodes $V = \{1, \ldots, n\}$ in the $\Gamma_1(n)$ graph is according to a topological order of the nodes.

Firstly, each two neighbouring poles of the edge-universal graph, p_{2s} and p_{2s+1} for $s \in \{1, \ldots, \lceil \frac{n}{2} \rceil\}$, are thought of as merged superpoles, with their fanin and fanout becoming 2. In a similar manner, any $G \in \Gamma_1(n)$ graph can be regarded as a $\Gamma_2(\lceil \frac{n}{2} \rceil)$ graph with supernodes, i.e. each pair $(2s, 2s+1)$ will be merged into one node in a $\Gamma_2(\lceil \frac{n}{2} \rceil)$ graph $G' = (V', E')$. If there are edges between the nodes in G, they are simulated with loops[1]. The set of edges of this graph G is partitioned to sets E_1 and E_2, s.t. $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are instances of $\Gamma_1(\lceil \frac{n}{2} \rceil)$ and $\Gamma_1(\lfloor \frac{n}{2} \rfloor)$, respectively. This can be done efficiently, as shown later in this section. The edges in E_1 are embedded as directed paths in Q, and the edges in E_2 as directed paths in R. Both E_1 and E_2 have at most one edge directed into and at most one directed out of any supernode and therefore, there is only one edge from E_1 and one from E_2 to be simulated going through any superpole in U_n as well. Thus, the edge coming into a superpole (p_{2s}, p_{2s+1}) in E_1 is embedded as a path through q_{s-1}, while the edge going out of the pole in E_1 is embedded as a path through q_s in the appropriate subgraph. Similarly, the edges in E_2 are simulated as edges through r_{s-1} and r_s. These paths can be chosen disjoint according to the induction hypothesis. Finally, the paths from q_{s-1} and r_{s-1} to superpole (p_{2s-1}, p_{2s}) as well as the paths from (p_{2s-1}, p_{2s}) to q_s and r_s can be chosen edge-disjoint due to the skeleton shown in Figs. 1a and b. With this, Valiant's graph construction is a valid edge-universal graph construction with asymptotically optimal size $\mathcal{O}(n \log n)$, and depth $\mathcal{O}(n)$ [Val76].

Valiant's Edge-Universal Graph Construction for $\Gamma_2(n)$ Graphs: Given a directed acyclic graph $G \in \Gamma_2(n)$, the set of edges E can be separated into two distinct sets E_1 and E_2, such that graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are instances of $\Gamma_1(n)$, having fanin and fanout 1 for each node [Val76]. Given the set of nodes $V = \{1, \ldots, n\}$, one constructs a bipartite graph $\overline{G} = (\overline{V}, \overline{E})$ with nodes $\overline{V} = \{m_1, \ldots, m_n, m'_1, \ldots, m'_n\}$ and edges $\overline{E}$ such that $(m_i, m'_j) \in \overline{E}$ if and only if $(i, j) \in E$. The edges of $\overline{G}$ and thus the corresponding edges of G can be colored in a way that the result is a valid two-coloring. Having fanin and fanout at most 2, such coloring can be found directly with the following method, used in the proof of Kőnig-Hall theorem in [LP09b]:

1: **while** There are uncolored edges in $\overline{G}$ **do**
2: Choose an uncolored edge $e = (m_i, m'_j)$ randomly and color the path or cycle that contains it in an alternating manner: the neighbouring edge(s) of an edge of the first color will be colored with the second color and vice versa.
3: **end while**

[1] We note that these G' graphs are constructed from the original $\Gamma_1(n)$ graph G in order to define the correct embedding. Therefore, they are not required to be acyclic.

This coloring can be performed in $\mathcal{O}(n)$ steps and it defines the edges in E_1 and E_2, s.t. E_1 contains the edges colored with color one and E_2 the ones with color two and $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ (cf. full version [KS16]).

With this method, the problem of constructing edge-universal graphs for $\Gamma_2(n)$ can be reduced to the $\Gamma_1(n)$ construction. After constructing two edge-universal graphs for $\Gamma_1(n)$ (i.e. $U_{n,1}$ and $U_{n,2}$), their poles are merged and an edge-universal graph for $\Gamma_2(n)$ is obtained. The merged poles now have fanin and fanout 2, since the poles of $U_{n,1}$ and $U_{n,2}$ previously had fanin and fanout 1. E_1 can then be edge-embedded using the edges of $U_{n,1}$ and E_2 using the edges of $U_{n,2}$.

Universal Circuits. We now describe how to construct UCs by means of Valiant's edge-universal graph construction for $\Gamma_2(n)$ graphs [Val76]. Our goal is to obtain an acyclic circuit built from special gates that simulate any acyclic Boolean circuit with u inputs, v outputs and k gates. In the circuit, the inputs of the gates are either connected to an input variable, to the output of another gate or are assigned a fixed constant. Due to the nature of Valiant's edge-universal graph construction, we have two restrictions on the original circuit. Firstly, all the gates must have at most two inputs and secondly, the fanout of inputs and gates must be at most 2, i.e., each input of the circuit and each output of any gate can only be the input of at most two later gates. This is necessary in order to guarantee that the graph of the original circuit has fanin and fanout 2. We note that the first restriction was present in case of the construction in [KS08b] as well, but the output of any input or any gate could be used multiple times. However, it was proven in [Val76] that the general case, where the fanout of the circuit can be any integer $m \geq 2$, can be transformed to the special case when $m \leq 2$ by introducing copy gates, where the resulting circuit will have k^* gates with $k \leq k^* \leq 2k + v$, where k denotes the number of gates and v the number of outputs in the circuit. We detail how this can be done in Sect. 4.1.

After this transformation, given a circuit C with u inputs, v outputs and k^* gates with fanin and fanout 2, the graph of C, denoted by G^C consists of a node for each gate, input and output variable and thus is in $\Gamma_2(u + v + k^*)$. The wires of circuit C are represented by edges in G^C. A *topological ordering* of the gates is chosen, which ensures that gate g_i has no inputs that are outputs of a later gate g_j, where $j > i$. The inputs and the outputs can be ordered arbitrarily within themselves as long as the inputs are kept before the topologically ordered gates and the outputs after them. Even though the output nodes cause an overhead in Valiant's UC, they are required to fully hide the topology of the circuit in the corresponding universal circuit. If, one can observe which gates provide the output of the computation, it might reveal information about the structure of the circuit, e.g. how many times is the result of an output gate used after being calculated. We ensure by adding nodes corresponding to the outputs that the last v nodes in U_{u+v+k^*} are the ones providing the outputs. We note that our understanding of universal circuits here slightly differs from Valiant's, since he constructs U_{u+k^*} [Val76].

Therefore, after obtaining G^C a Γ_2 edge-universal graph U_{u+v+k^*} is constructed, into which G^C is edge-embedded. Valiant shows in [Val76] how to obtain the universal circuit corresponding to U_{u+v+k^*} and how to program it according to the edge-embedding of G^C. Firstly, the first u poles become inputs, the next k^* poles are so-called universal gates, and the last v poles are outputs in the universal circuit. A *universal gate* denoted by $U(\mathrm{in}_1, \mathrm{in}_2; c_0, c_1, c_2, c_3)$, can compute any function with two inputs in_1 and in_2 and four control bits c_0, c_1, c_2 and c_3 as in Eq. 1.

$$\mathrm{out}_1 = c_0\overline{\mathrm{in}_1\mathrm{in}_2} \oplus c_1\overline{\mathrm{in}_1}\mathrm{in}_2 \oplus c_2\mathrm{in}_1\overline{\mathrm{in}_2} \oplus c_3\mathrm{in}_1\mathrm{in}_2. \tag{1}$$

The rest of the nodes of the edge-universal graph are translated into *universal switches* or *X gates*, denoted by $(\mathrm{out}_1, \mathrm{out}_2) = X(\mathrm{in}_1, \mathrm{in}_2; c)$ that are defined by one control bit c and return the two input values either in the same or in reversed order as in Eq. 2.

$$\mathrm{out}_1 = \overline{c}\,\mathrm{in}_1 \oplus c\,\mathrm{in}_2, \qquad \mathrm{out}_2 = c\,\mathrm{in}_1 \oplus \overline{c}\,\mathrm{in}_2. \tag{2}$$

The programming of the universal circuit means specifying the control bit of each universal switch and the four control bits of each universal gate. The universal gates are programmed according to the simulated gates in C and the universal switches according to the paths defined by the edge-embedding of the graph of the circuit G^C in the edge-universal graph U_{u+v+k^*}. Depending on if the path takes the same direction during the embedding (e.g. arrives from the left and continues on the left) or changes its direction at a given node (e.g. arrives from the left and continues on the right), the control bit of the universal switch can be programmed accordingly. In Sect. 4.1, we detail our concrete method for programming the universal circuit and discuss efficient implementations of universal gates and switches.

2.2 Universal Circuit Construction from [KS08b]

The universal circuit construction from [KS08b] is built from three main building blocks (cf. full version [KS16]) that we summarize in this section. The construction uses efficient building blocks for hiding the wiring of the u inputs and v outputs, using the fact that the maximum number of inputs to a circuit with k gates is $2k$ and the maximum number of outputs is k. A recursive building block with size $\mathcal{O}(k \log^2 k)$ is constructed for hiding the wiring between the gates.

For hiding the input wiring, a *selection block* $S^u_{2k \geq u}$ is used, i.e., a programmable block that selects for $2k$ outputs one of $u \leq 2k$ inputs. This means that with the u inputs of circuit C, it can be programmed to assign the output wires according to the original structure of C and assign duplicates to the rest of the wires. The authors show an efficient implementation of selection blocks with size $\mathcal{O}(k \log k)$ and depth $\mathcal{O}(k)$ with a small constant factor [KS08b].

For hiding the output wiring, the authors use a smaller selection block. We note that the usage of their so-called *truncated permutation block* is enough to

program the output wires according to the original topology of C as no duplicates can occur. This truncated permutation block $TP_v^{k \geq v}$ permutes a subset of the maximal k inputs to the $v \leq k$ outputs. An efficient construction of size $\mathcal{O}(k \log v)$ and depth $\mathcal{O}(\log k)$ is given in [KS08b].

A *universal block* UB_k is placed between the input selection block and the output permutation block. It takes care of the simulation of the gates using universal gates and ensures that each possible wiring can be implemented in the UC. The universal block construction is recursive, makes use of two universal blocks of smaller size with a selection block and a mixing block (essentially a layer of universal switches with one output) in between them. The $\mathcal{O}(k \log^2 k)$ size of this universal block is asymptotically not optimal and its $\mathcal{O}(k \log k)$ depth is also a factor of $\log k$ larger than Valiant's UC's. Thus, despite the efficiency of the other two building blocks, the construction from [KS08b] yields larger circuits than Valiant's UC in most cases. However, we note that using some of its building blocks can be beneficial in some scenarios (cf. Sect. 3.2).

3 The Size and the Depth of Valiant's Construction

In this section, we obtain new formulae for the size and the depth of Valiant's construction: the Γ_1 edge-universal graph construction is described in Sect. 3.1 and the universal circuit construction in Sect. 3.2. The *size of the edge-universal graph* is the number of nodes, counting all the poles and nodes created while using Valiant's construction. The *depth of the edge-universal graph* is the number of nodes on the longest path between any two nodes. When considering UCs and the PFE application, since XOR gates can be evaluated for free in secure computation [KS08a], the *ANDsize of the universal circuit* is the number of AND gates that are needed to realize the UC in total. The *ANDdepth of the universal circuit* in this scenario is the maximum number of AND gates between any input and output. For the sake of generality, we give the *total size* and *depth* of Valiant's UC construction with respect to both the AND and XOR gates that are used. Our implementation of universal gates and switches is optimized for PFE (cf. Sect. 4.1) and therefore uses the fewest AND gates possible. However, the total size and depth can be relevant when optimizing for other applications, in which case our implementation gives an upper bound that can be improved. For instance, when XOR and AND gates have the same costs, one needs to minimize the total number of gates instead of the number of AND gates as in [LMS16].

3.1 The Size and the Depth of the Γ_1 Edge-Universal Graph

In the skeleton, node A in Fig. 1a is redundant, since one can choose to embed the edge $(y, n-1)$ as (p_y, p_{n-1}) through Q, and (z, n) as (p_z, p_n) through R for any y and z nodes [Val76]. Thus, the number of nodes other than poles $\textsc{Exact}(n)$, for even n becomes

$$\textsc{Exact}(n) = 2 \cdot \textsc{Exact}\left(\frac{n-2}{2}\right) + 5 \cdot \frac{n-2}{2}. \tag{3}$$

For odd n, the construction makes use of $\frac{n-1}{2}$ poles in Q and $\frac{n-3}{2}$ poles in R. Then, edge (y, n) is embedded as (p_y, p_n) through Q for any y node, and node A is again redundant. Thus,

$$\textsc{Exact}(n) = \textsc{Exact}\left(\frac{n-1}{2}\right) + \textsc{Exact}\left(\frac{n-3}{2}\right) + 5 \cdot \frac{n-3}{2} + 3. \tag{4}$$

Using these recursive formulae, given the value n, it is possible to obtain the exact number of nodes other than poles in U_n. Valiant includes optimizations for starting off the recursion: for $1, 2, 3, 4, 5$ and 6 nodes; the respective number of additional nodes are $0, 0, 0, 3, 7$ and 9 (cf. Figs. 1c, d and e). Thus, a simple algorithm using dynamic programming based on the recursion relations of Eqs. 3 and 4 yields the exact number of nodes other than the original n poles that are created during the edge-universal graph construction. It depends on the parity of the input n at each iteration and unfortunately does not yield a closed formula for the size of Valiant's edge-universal graph construction, which is $n + \textsc{Exact}(n)$.

Valiant states that using his method, an edge-universal graph for $\Gamma_1(n)$ can be found "*with fewer than* $\frac{19}{8} n \log_2 n$ *nodes, and fanin and fanout 2*" [Val76]. As mentioned in Sect. 2.1, we consider the more detailed algorithm that yields the result with a slightly larger prefactor of $2.5n \log_2 n$ instead of $2.375n \log_2 n$. In this section, we sharpen this bound and give an approximate closed formula for the size of the construction. We first give upper and lower bounds, and then derive an approximation for a closed formula. For our lower bound, we consider the case when only the formula for even numbers, i.e., Eq. 3, is considered. This yields our lower bound of

$$n + 5 \left(\sum_{i=0}^{\log_2 n - 1} 2^i \left(\frac{n}{2^{i+1}} - \frac{2(2^{i+1} - 1)}{2^{i+1}} \right) \right) = 2.5n \log_2 n - 9n + 5 \log_2 n + 10. \tag{5}$$

The upper bound can be obtained similarly, considering the case when only the formula for odd numbers with $5 \cdot \left(\frac{n-1}{2}\right)$ is considered

$$n + 5 \left(\sum_{i=0}^{\log_2 n - 1} 2^i \left(\frac{n}{2^{i+1}} - \frac{2^{i+1} - 1}{2^{i+1}} \right) \right) = 2.5n \log_2 n - 4n + 2.5 \log_2 n + 5. \tag{6}$$

Figure 2 depicts our upper and lower bounds along with Valiant's upper bound on the same construction for up to 100 000 nodes. We observe that the mean of our bounds is very close to the exact number of nodes. Figure 3 shows that already after a couple of hundreds of poles, it only slightly deviates from the exact number of nodes $\textsc{Exact}(n)$. Thus, we accept

$$\text{size}(U_n) \approx 2.5n \log_2 n - 6.5n + 3.75 \log_2 n + 7.5 \tag{7}$$

as a good approximation of the closed formula for the size of the construction, noting that an estimated deviation of at most 2 % compared to the exact number of nodes, i.e., $\varepsilon \leq 0.02 \cdot \text{size}(U_n)$ may occur.

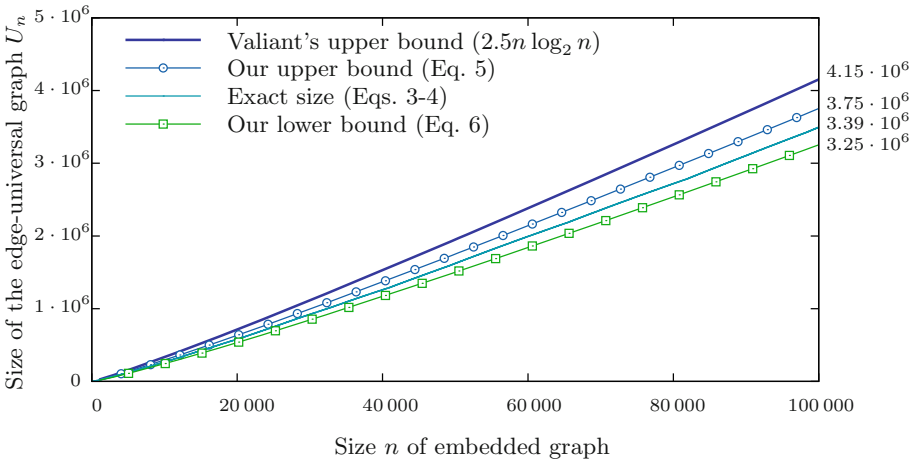

Fig. 2. Our upper and lower bounds for the size of Valiant's edge-universal graph construction for $\Gamma_1(n)$ graphs, along with Valiant's upper bound on the same construction and the exact size $\textsc{Exact}(n)$, considering the size of the embedded graph $n \in \{1, \ldots, 100\,000\}$ (Color figure online).

The depth of the edge-universal graph, i.e., the maximum number of nodes between any two nodes is defined by the number of nodes between p_1 and p_n in the skeleton (cf. Figs. 1a and b). Thus, $\text{depth}(U_n) = 3n - 3$ for even n and $\text{depth}(U_n) = 3n - 2$ for odd n.

3.2 The Size and the Depth of Valiant's Universal Circuit

As described in Sect. 2.1, a universal circuit is constructed by means of an edge-universal graph for graphs with fanin and fanout 2, which is in turn constructed from two Γ_1 edge-universal graphs with poles merged together and thus taken only once into consideration. When constructing a UC, the number of inputs u, the number of outputs v and the number of gates k is public. We set k^* as the number of gates in the equivalent fanout-2 circuit, where $k \leq k^* \leq 2k + v$, in order to be able to later fairly compare with the UC construction of [KS08b]. We consider k^* as the public parameter instead of k, since without the knowledge of the original number of simulated gates, it does not reveal information about the simulated circuit. If the original k is public, one can hide k^* by setting it to its maximal value $2k + v$. Thus, using Valiant's UC construction, a Γ_2 edge-universal graph with $u + v + k^*$ poles is constructed and thus, our approximative formula for the size of the Γ_2 edge-universal graph corresponding to the graph of the circuit would become $2 \cdot \text{size}(U_{u+v+k^*}) - (u + v + k^*)$ and the exact number would be $u + v + k^* + 2 \cdot \textsc{Exact}(u + v + k^*)$, i.e., the $u + v + k^*$ merged poles of the two edge-universal graphs plus the exact number of nodes other than poles. Therefore, the size of Valiant's UC is

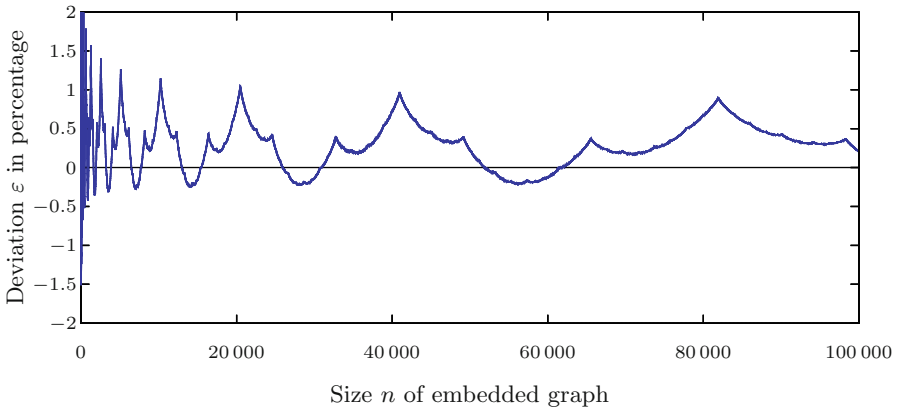

Fig. 3. The deviation of the mean of our upper and lower bounds (Eqs. 5 and 6) from the exact size of the edge-universal graph $\textsc{Exact}(n) + n$, considering the size of the embedded graph $n \in \{1, \ldots, 100\,000\}$.

$$\begin{aligned}\text{size}(UC^{\text{Valiant}}_{u,v,k^*}) \approx &[5(u+v+k^*)\log_2(u+v+k^*) - 15(u+v+k^*) \\ &+ 7.5\log_2(u+v+k^*) + 15] \cdot \text{size}(X) + k^* \cdot \text{size}(U)\end{aligned} \tag{8}$$

and the depth stays

$$\text{depth}(UC^{\text{Valiant}}_{u,v,k^*}) \approx [2(u+v+k^*) - 2] \cdot \text{depth}(X) + k^* \cdot \text{depth}(U). \tag{9}$$

When transforming the Γ_2 edge-universal graph into a UC, the first u poles are associated with inputs, the last v poles with outputs, and the k^* poles between are realized with universal gates (cf. Eq. 1) and their programming is defined by the corresponding gates in the simulated circuit. The rest of the nodes of the edge-universal graph are translated into universal switches (cf. Eq. 2), whose programming is defined by the edge-embedding of the graph of the circuit into the Γ_2 edge-universal graph. Thus, the size and depth of Valiant's UC can be directly derived from the size of the Γ_2 edge-universal graph. However, we include two optimizations to obtain a smaller size of the UC. The first optimization improves already the size of the edge-universal graph and the second optimization is applied when translating the edge-universal graph into a UC description (cf. Sect. 4.1).

1. **Optimization for Input and Output Nodes:** We observe that obviously circuit inputs need no ingoing edges and circuit outputs need no outgoing edges. Therefore, since u, v and k^* are publicly known, we optimize by deleting nodes that become redundant while canceling the edges going to the first u (input) and coming from the last v (output) nodes. Depending on the parity of u, v and $u + v + k^*$, the number of redundant switching nodes is $u+v-3\pm1$ in both Γ_1 edge-universal graphs that build up the graph of the UC. Therefore, we have, on average, $2(u + v - 3)$ redundant nodes, which number we use in our calculations further on. This optimization also affects the depth by, on average, $u + v - 3$.

2. **Optimization for Fanin-1 Nodes:** We observe that in the skeleton of the Γ_1 edge-universal graph construction there is a fanin-1 node (denoted with B in Figs. 1a and b). Such fanin-1 nodes exist in the base-cases for a small number of poles as well (cf. Figs. 1c, d and e). These nodes are important to achieve fanin and fanout 2 of each nodes in the graph, but can be ignored and replaced with wires when translated into a circuit description, essentially resulting in the same UC. According to Valiant's construction, these gates would translate into universal switches with one real input (and an other arbitrary one). Instead, we translate each of them into two wires and therefore set the second input to the same as the first one. Since at least one such node can be ignored in each subgraph when nodes are translated into gates, this results in altogether around

$$2 \cdot \left(\sum_{i=0}^{\log_2(u+v+k^*)-1} 2^i \right) - 1 = 2(u+v+k^*) - 3 \tag{10}$$

less gates for the two Γ_1 edge-universal graphs. This improvement has no effect on the depth of the construction.

Since both the size and the depth are dependent on the underlying representation of the circuit building blocks (of the universal gate U and of the universal switch or X gate), and the secure computation protocol, we express the size of the universal circuit with the size and depth of U and of X as parameters. Including the above optimizations of altogether $4(u+v) + 2k^* - 9$, the approximate formula for the size of Valiant's optimized UC construction becomes

$$\begin{aligned} \text{size}(UC^{\text{opt}}_{u,v,k^*}) \approx & [5(u+v+k^*)\log_2(u+v+k^*) - 17k^* - 19(u+v) \\ & + 7.5\log_2(u+v+k^*) + 24] \cdot \text{size}(X) + k^* \cdot \text{size}(U). \end{aligned} \tag{11}$$

To obtain the exact size of the UC, we use the recursive relations depicted in Eqs. 3 and 4 and include our optimizations. Thus, we obtain

$$\begin{aligned} \text{size}_{\text{exact}}(UC^{\text{opt}}_{u,v,k^*}) = & [2 \cdot \textsc{Exact}(u+v+k^*) \\ & - 4(u+v) - 2k^* + 9] \cdot \text{size}(X) + k^* \cdot \text{size}(U). \end{aligned} \tag{12}$$

From the depth of the edge-universal graph, the depth of the UC becomes

$$\text{depth}(UC^{\text{opt}}_{u,v,k^*}) \approx [u+v+2k^*+3] \cdot \text{depth}(X) + k^* \cdot \text{depth}(U). \tag{13}$$

Depending on the application, size(X) and size(U) as well as depth(X) and depth(U) can be optimized. Due to the PFE application, where XOR gates can be evaluated for free, we assess the ANDsize and ANDdepth of our AND-optimized implementations of universal gates and switches (cf. Sect. 4.1). In general, a universal gate can be realized with 3 AND gates (and 6 XOR gates), and ANDdepth of 2 (total depth of 6). Universal switches can be realized with only one AND gate (and 3 XOR gates), and ANDdepth of 1 (total depth of 3) [KS08a].

For private function evaluation, the size and the depth of U can be further optimized depending on the underlying secure computation protocol. In case the SFE implementation uses Yao's garbled circuit protocol [Yao86], both ANDsize(U) and ANDdepth(U) can be minimized to 1, due to the fact that in some garbling schemes the evaluator does not learn the type of the evaluated gate such as in case of garbled 3-row-reduction [NPS99]. Therefore, a universal gate can be implemented with one 2-input non-XOR gate [PSS09].

Optimized Hybrid Universal Circuit Construction: We investigate if hybrid methods utilizing building blocks of both UC constructions, i.e., of both [Val76] summarized in Sect. 2.1 and [KS08b] in Sect. 2.2, could yield better size. The simulation of the k gates of the original circuit is asymptotically more efficient using Valiant's UC construction due to the logarithmic factor, despite the overhead caused by taking the equivalent fanout-2 circuit with k^* gates, where $k \leq k^* \leq 2k + v$. However, we calculate if the modular approach of [KS08b] using a selection block $S^u_{m \geq u}$ for selecting the input variables or a truncated permutation block $TP_v^{k^* \geq v}$ for the output variables results in a smaller size.

Placing a selection block on top of Valiant's UC with m universal gates would imply a selection block $S^u_{m \geq u}$ which is then programmed to direct the u inputs of the circuit to the proper inputs of the m universal gates. Depending on how the output nodes are represented, m is either $2(k^* + v)$ for the case when including the outputs in Valiant's construction or $2k^*$ for the construction with a truncated permutation block. In the latter case, $TP_v^{k^* \geq v}$ takes care of permuting a subset of the outputs of the k^* gates, resulting in the v outputs of the UC. A selection block $S^u_{m \geq u}$ has size $\frac{u+m}{2} \log_2 u + m \log_2 m - u + 1$ and depth $2 \log_2 u + 2 \log_2 m + m - 2$, and a truncated permutation block $TP_v^{k^* \geq v}$ has size $\frac{k^*+v}{2} \log_2 v - 2v + k^* + 1$ and depth $\log_2 k^* + \log_2 v - 1$ [KS08b] (cf. full version [KS16]).

Let us take three scenarios into consideration, depending on the number of inputs u and the number of outputs v. The number of gates in the circuit to be simulated is k and the number of gates in the equivalent fanout-2 circuit is k^* with $k \leq k^* \leq 2k + v$.

1. **Constant I/O Case: $u = c_1$ constant, $v = c_2$ constant:** If both u and v are constant values c_1 and c_2 respectively, as is the case in many applications that compute a non-trivial function with relatively few inputs and outputs, the size of the selection block becomes $\approx 2k^* \log_2 k^* + (2 + \log_2 c_1)k^*$ and the size of the truncated permutation block is $\approx (0.5 \log_2 c_2 + 1)\, k^*$. With Valiant's UC construction, the overhead caused by a constant number of inputs and outputs is around $5(c_1 + c_2) \log_2 k^*$. The depth of Valiant's UC is only affected with constant overhead, while the depth of the selection and permutation blocks are $\approx 2k^* + 2 \log_2 k^*$ and $\approx \log_2 k$, respectively. Thus, both for the inputs and the outputs, Valiant's UC is an asymptotically better solution in the case with a constant number of inputs and outputs.

2. Many Inputs: $u \sim k$, $v = c$ **constant:** For many inputs where u is around the number of gates k and we have a constant number of c outputs, we include these c nodes in Valiant's UC instead of using a truncated permutation block due to the same reasoning as in the previous case. However, a selection block can be constructed to direct k inputs to $k^* + c$ universal gates. Thus, its size becomes $\approx 2k^* \log_2 k^* + k^* \log_2 k + 0.5k \log_2 k + 2k^* - k + 3c \log_2 k^*$ and its depth $\approx 2k^* + 2\log_2 k^* + 2\log_2 k$. In case of Valiant's UC construction, k inputs result in an overhead of $\approx 5k \log_2 k - 9k + 5c \log_2 k$ for the size and $\approx k$ for the depth, since a large part (up to a half) of the circuit is built in order to hide the input wiring. Therefore, in this scenario it is often worth to use a hybrid method, utilizing the selection block from [KS08b] for input selection. Our *many inputs hybrid* construction places a selection block on top of a UC with $k^* + c$ universal gates and has approximate size when $u \sim k$ and v is constant c

$$\begin{aligned}\text{size}(UC^{\text{many I}}_{k,c,k^*}) \approx [&7k^* \log_2 k^* + k^* \log_2 k + 0.5k \log_2 k - k - 15k^* \\ &+ (7.5 + 5c) \log_2 k^* + 3c \log_2 k^* + \mathcal{O}(1)] \cdot \text{size}(X) + k^* \cdot \text{size}(U)\end{aligned} \tag{14}$$

and approximate depth

$$\begin{aligned}\text{depth}(UC^{\text{many I}}_{k,c,k^*}) \approx [&4k^* + 2\log_2 k^* + 2\log_2 k + \mathcal{O}(1)] \cdot \text{depth}(X) \\ &+ k^* \cdot \text{depth}(U).\end{aligned} \tag{15}$$

3. Maximal I/O Case: $u \sim 2k$, $v \sim k$**:** For circuits with $u \sim 2k$ inputs and $v \sim k$ outputs, we discuss the possibility of using both an input selection block and an output permutation block. The size of the selection block is $\approx 2k^* \log_2 k^* + k^* \log_2 k + k \log_2 k + 3k^* - k$ and its depth becomes $\approx 2k^* + 2\log_2 k^* + 2\log_2 k$, which is more beneficial (when it comes to the size) than the $\approx 10k \log_2 k - 12k$ size overhead and $\approx 2k$ depth overhead in Valiant's construction caused by $2k$ inputs (up to half of the UC is constructed for inputs only). The truncated permutation block has size $\approx 0.5k^* \log_2 k + 0.5k \log_2 k + k^* - 2k$ and depth $\approx \log_2 k^* + \log_2 k$, while the same amount of outputs in Valiant's construction introduces at least $5k \log_2 k - 9k$ new switches with depth of $\approx k$. Thus, for the case when the maximal $2k$ inputs and k outputs are considered, we conclude that it is advantageous to use our *maximal I/O hybrid* construction, utilizing Valiant's graph construction for the k^* gates [Val76], a selection block for the inputs and a truncated permutation block for the outputs [KS08b]. This yields an approximate size when $u \sim 2k$ and $v \sim k$

$$\begin{aligned}\text{size}(UC^{\text{max I/O}}_{2k,k^*,k}) \approx [&7k^* \log_2 k^* + 1.5k^* \log_2 k + 1.5k \log_2 k - 13k^* - 3k \\ &+ 7.5 \log_2 k^* + \mathcal{O}(1)] \cdot \text{size}(X) + k^* \cdot \text{size}(U)\end{aligned} \tag{16}$$

and an approximate depth

$$\begin{aligned}\text{depth}(UC^{\text{max I/O}}_{2k,k^*,k}) \approx [&4k^* + 3\log_2 k^* + 3\log_2 k + \mathcal{O}(1)] \cdot \text{depth}(X) \\ &+ k^* \cdot \text{depth}(U).\end{aligned} \tag{17}$$

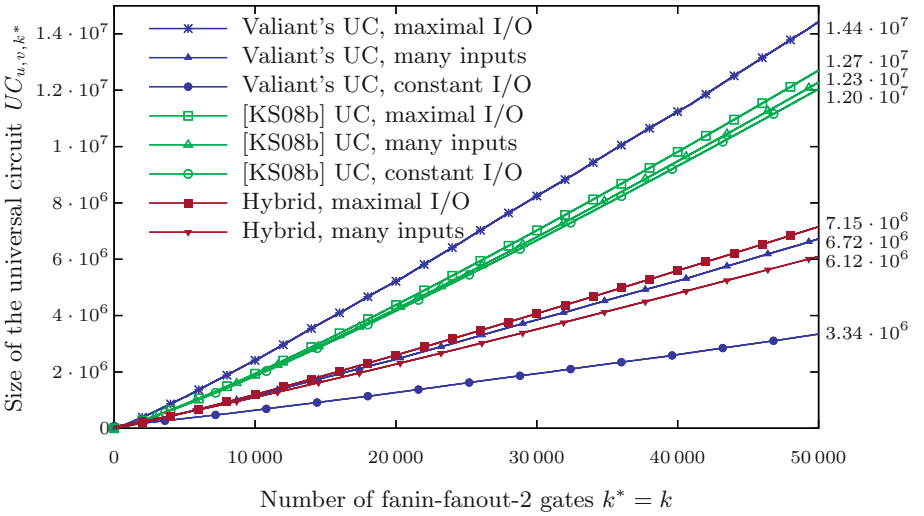

Fig. 4. Comparison between the sizes of the universal circuit constructions for $k^* = k \in \{0, \ldots, 50\,000\}$ gates, considering the three scenarios: *constant I/O* with constant number of inputs and outputs, *many inputs* with $\sim k$ inputs and constant outputs and *maximal I/O* with $\sim 2k$ inputs and $\sim k$ outputs (Color figure online).

We conclude that in case of a large number of inputs and outputs it is beneficial to construct a hybrid UC, making use of both existing constructions (cf. Sects. 2.1 and 2.2). Most practical applications have input and output with constant size and only some specific applications use input size linear in the number of gates (e.g. simple computations on large databases). Thus, we consider Valiant's construction as the most beneficial for general purposes, however we have shown, that one can optimize the construction for many inputs or outputs by adding selection or truncated permutation blocks from [KS08b].

Comparison with the Universal Circuit Construction from [KS08b]. In [KS08b], a universal circuit construction was proposed with approximate size $1.5k \log_2^2 k + 2.5k \log_2 k$. This was calculated with the doubled size of the universal switches, not yet considering the free-XOR optimizations of [KS08a]. We recalculated the size of the construction with our additional optimization for the outputs described in Sect. 2.2. We give our detailed calculations in the full version [KS16] and summarize its exact size here as

$$\begin{aligned}\text{size}(UC^{[\text{KS08b}]}_{u,v,k}) = [&0.75k \log_2^2 k + 2.25k \log_2 k + (0.5 + k) \log u + \\ &(0.5k + 0.5v) \log v + 5k - u - 2v] \cdot \text{size}(X) + k \cdot \text{size}(U),\end{aligned} \tag{18}$$

and from [KS08b] we know that its depth is

$$\begin{aligned}\text{depth}(UC^{[\text{KS08b}]}_{u,v,k}) = [&k \log_2 k + 2k + 7 \log_2 k + 2 \log_2 u + \\ &\log_2 v - 14] \cdot k \cdot \text{depth}(U).\end{aligned} \tag{19}$$

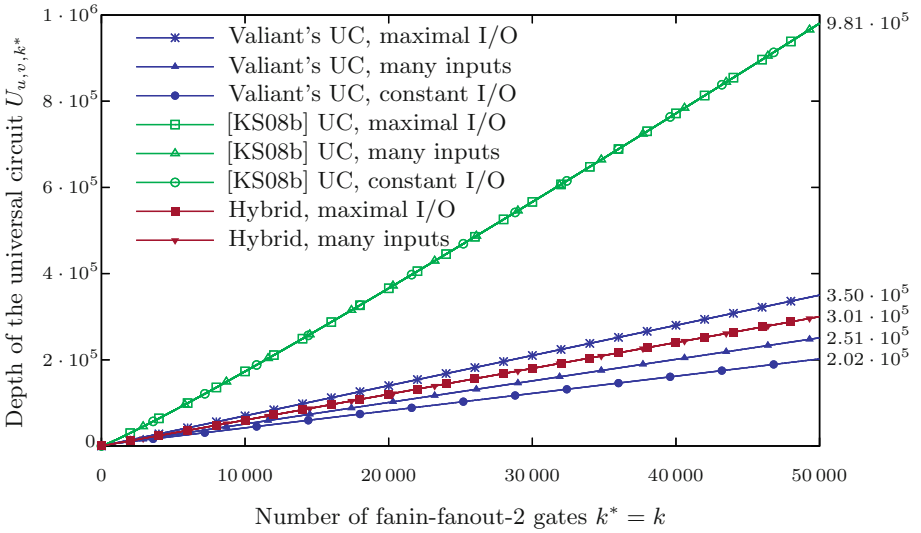

Fig. 5. Comparison between the depths of the universal circuit constructions for $k^* = k \in \{0, \ldots, 50\,000\}$ gates, considering the three scenarios: *constant I/O* with constant number of inputs and outputs, *many inputs* with $\sim k$ inputs and constant outputs and *maximal I/O* with $\sim 2k$ inputs and $\sim k$ outputs (Color figure online).

It was concluded in [KS08b] that this construction outperforms Valiant's construction for circuits with up to 5 000 gates. However, this was achieved using the assumption that Valiant's UC has size $\approx 9.5(u + 2v + 2k) \log_2(u + 2v + 2k)$, which can vary between two to four times its actual size. On the one hand, a factor of two of this difference is due to the free-XOR optimizations in [KS08a]. On the other hand, [KS08b] used the maximal $k^* = 2k+v$ in their approximation. In Sect. 4.2, we show on concrete example circuits that k^* stays significantly below this upper bound. The construction described in detail in Sect. 2.1 has a larger constant factor 5, but due to the logarithmic factor it outperforms the construction from [KS08b] (Sect. 2.2) already for a few hundred gates in the constant I/O case. Figures 4 and 5 compare the sizes and depth of the different UC constructions, respectively in the three scenarios described above, with the lowest possible gate number $k^* = k$. When considering the hybrid approach, the method corresponding to the given scenario is indeed always the most efficient construction for many inputs and/or outputs. We give a comparison for the upper bound case $k^* = 2k+v$, for the sizes of all universal circuit constructions for well-known circuits from [TS15] and compare their structure in the full version [KS16].

4 Implementing Valiant's Universal Circuit in Practice

In this section, we detail the challenges that we faced while demonstrating the practicality of Valiant's universal circuit construction. We show how to construct

a universal circuit from a standard circuit description and how to program it accordingly. We validate our results with an implementation, creating a novel toolchain for private function evaluation, using two existing frameworks as frontend and backend of our application. We emphasize that our tool for constructing and programming UC is generic and can easily be adapted to other secure computation frameworks or other applications of UCs listed in Sect. 1.2.

4.1 Our Tool for Universal Circuit Construction and Toolchain for Private Function Evaluation

The architecture of our toolchain for PFE using universal circuits is shown in Fig. 6. In this section, we describe its different artifacts and its use of the Fairplay [MNPS04] and ABY [DSZ15] frameworks. Our implementation is available online at http://encrypto.de/code/UC.

Step 1. Compiling Input Circuits from High-Level Functionality: Due to its easy adoptability, we decided to use the Fairplay compiler [MNPS04, BNP08] with the FairplayPF extension [KS08b] to translate the functionality described in the high-level SFDL format to the Fairplay circuit description called Secure Hardware Definition Language (SHDL). The FairplayPF extension already converts circuits with gates of an arbitrary fanin into gates with at most two inputs, which is required for Valiant's construction as well. However, in case of Valiant's UC construction, there is another restriction on the input circuit. It has to have fanout 2, i.e., the outputs of all the gates and inputs can only be used as the input of at most two later gates.

In case the input circuit does not follow this restriction, an algorithm places a binary tree in place of each gate with fanout larger than 2, following Valiant's proposition: *"Any gate with fanout $x+2$ can be replaced by a binary fanout tree with $x+1$ gates"* [Val76, Corollary 3.1]. This is done using so-called *copy gates*, i.e., identity gates, each of them eliminating one from the extra fanout of the original gate. An upper bound can be given on the number of copy gates. The class of Boolean functions with u inputs and v outputs that can be realized by acyclic circuits with k gates and arbitrary fanout, can also be realized with an acyclic fanout-2 circuit with $k \leq k^* \leq 2k+v$ gates [Val76, Corollary 3.1]. We give concrete examples in Sect. 4.2 on how this conversion changes the input circuit size for practical circuits and show that in most cases, the resulting number of gates remains significantly below the upper bound $2k+v$.

Step 2. Obtaining the Γ_2 Graph of the Circuit: From the SHDL description of a C circuit with fanin and fanout 2, the Γ_2 graph G^C of the circuit C can be directly generated as described in Sect. 2.1: with the number of inputs u, the number of outputs v and the number of gates k^* in circuit C, G^C has $u+v+k^*$ nodes and the wires are represented as edges in the graph. Then, the first u nodes in the topological order correspond to the inputs, the last v nodes to the outputs and the nodes in between them to the k^* ordered gates. We note that since C had fanin and fanout 2, the resulting G^C graph is in $\Gamma_2(u+v+k^*)$.

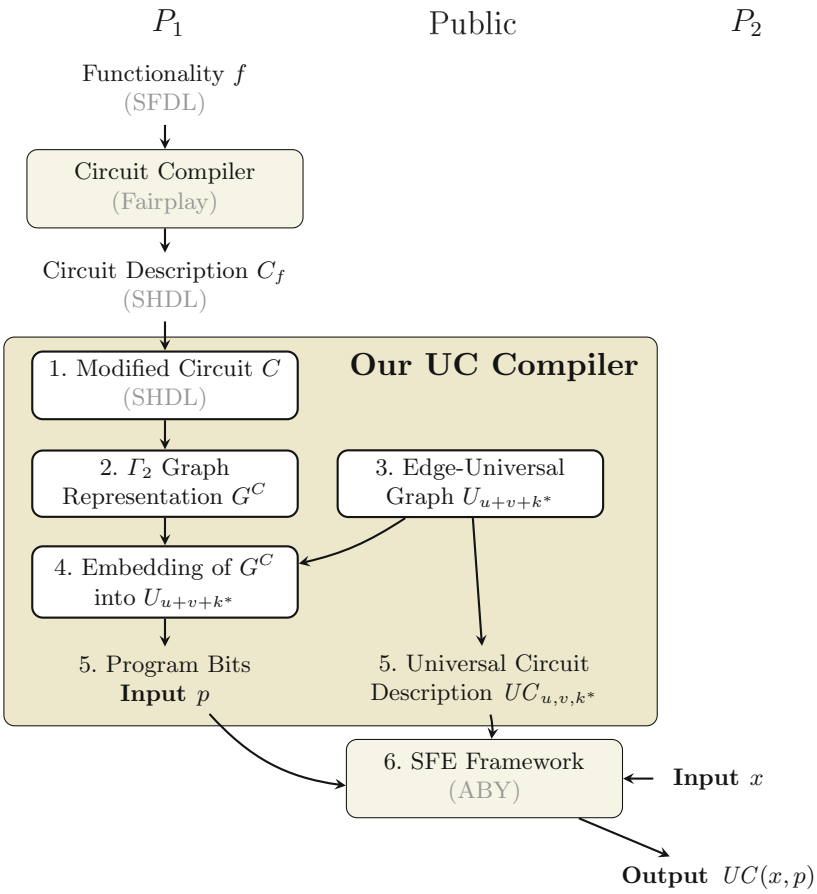

Fig. 6. Our toolchain for universal circuits and private function evaluation.

Therefore in G^C, each node can have at most two incoming edges, one defined to be the first and the other the second. It is possible in the modified SHDL circuit description that an internal value becomes two times the first or two times the second input of gates. This is due to the fact that in the original SHDL circuit with arbitrary fanout, a value could be the input of arbitrary number of later gates. Transforming the circuit to a fanout-2 circuit by adding copy gates allows a value to be an input only two times, but the order of the inputs is fixed. Therefore, in such a case when a value is the second time the same input to a gate (i.e., first or second), besides the two inputs, the two middle bits of the function table of the gate must be reversed as well (i.e., to compute $f(\text{in}_1, \text{in}_2)$ instead of $f(\text{in}_2, \text{in}_1)$) for the correct programming of the universal circuit in Step 5.

Step 3. Generating Γ_2 Edge-Universal Graph U_n: Knowing the number of input bits u, the number of gates k^* and the number of output bits v one can construct the corresponding edge-universal graph U_n, where $n = u + v + k^*$, with out input-output optimization from Sect. 3.2. We note that no knowledge is necessary about the topology or the gate tables in circuit C for this step. As we described in Sect. 2.1, two edge-universal graphs for $\Gamma_1(n)$, i.e. $U_{n,1}$ and $U_{n,2}$, are merged in order to obtain an edge-universal graph for $\Gamma_2(n)$, such that the poles are merged and the edges coming into and going out from them become as follows: the edges in $U_{n,1}$ will be the first input and output for each pole, the edges in $U_{n,2}$ will be the second input and output. For efficiency reasons, we directly generate the merged edge-universal graph, i.e., an edge-universal graph for $\Gamma_2(n)$, with the poles as common nodes.

We include our optimization for the input and output nodes from Sect. 3.2 and Valiant's optimizations for $n \in \{2, 3\}$, but do not consider Valiant's optimizations for $n \in \{4, 5, 6\}$ (cf. Figs. 1c, d, and e). These special cases lead to a specific edge-embedding for the nodes and result in linear improvement only in very rare cases. Moreover, with our second optimization from Sect. 3.2, we ignore most of the extra nodes when the graph is translated into a universal circuit description, i.e., we have for $n = \{4, 5, 6\}$ only $\{3, 5, 8\}$ additional nodes other than poles, respectively, in our implementation which is already an improvement over Valiant's original optimizations.

We note that the edge-universal graph (with undefined function tables and control bits for the universal switches) can be publicly generated. However, the party programming it has to either generate or receive a copy of it for programming the control bits according to the topology of the simulated circuit (i.e., to edge-embed G^C into U_n).

Step 4. Programming U_n According to an Arbitrary $\Gamma_2(n)$ Graph: The Γ_2 graph of the circuit G^C with n nodes is partitioned into two $\Gamma_1(n)$ graphs G_1^C and G_2^C which are embedded into the two edge-universal graphs for $\Gamma_1(n)$ that build up U_n. Valiant proved in [Val76] that for any topologically ordered $\Gamma_1(n)$ graph, for any $(i, j) \in E$ and $(k, l) \in E$ edges there exist edge-disjoint paths in U_n between the i^{th} and the j^{th} poles and between the k^{th} and the l^{th} poles. We described Valiant's method in Sect. 2.1 and here we show the algorithm that uniquely defines these paths in U_n.

For the description of our algorithm, we first define a $\Gamma_1(n)$ supergraph, which is a $\Gamma_1(n)$ graph with additionally a binary tree of Γ_1 graphs of decreasing size. These Γ_1 graphs uniquely define the embedding of the edges into U_n. When embedding an edge (i, j) of the topologically ordered graph G into the edge-universal graph, one needs to construct the supergraph of G as described in Algorithm 1 and then look at the binary tree in the supergraph. The path of the edge (i, j) defines the edge-embedding uniquely. This means that if edge $(\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil - 1)$ is in the left subgraph of G, then it can be embedded through subgraph Q in U_n, otherwise it is in the right subgraph of G and can be embedded through subgraph R in U_n. The unique embedding happens through $\{r_{\lceil \frac{i}{2} \rceil}, r_{\lceil \frac{j}{2} \rceil - 1}\}$ or through $\{q_{\lceil \frac{i}{2} \rceil}, q_{\lceil \frac{j}{2} \rceil - 1}\}$, utilizing the unique shortest path between them, through subpoles further identified by smaller subgraphs of G.

Algorithm 1. SUPERGRAPH(G)

Input: $\Gamma_1(n)$ graph G with set of nodes $V = \{1, \ldots, n\}$
Output: $\Gamma_1(n)$ supergraph

Create a graph H with $\lceil \frac{n}{2} \rceil - 1$ nodes ▷ H Γ_2 graph (with possible loops)
if there exist an edge (i, j) in G **and** $\lceil \frac{j}{2} \rceil - 1 \geq \lceil \frac{i}{2} \rceil$ **then**
    Add edge $\left(\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil - 1\right)$ in H ▷ each pair of nodes in G is one node in H
end if
Partition H into two Γ_1 graphs G_1 of size $\lceil \frac{n}{2} \rceil - 1$ and G_2 of size $\lfloor \frac{n}{2} \rfloor - 1$ using Kőnig's theorem as in §2.1
    ▷ in odd case, the $(e, \lceil \frac{n}{2} \rceil - 1)$ edge in H for arbitrary e will be added in G_1
if size(G_1) $\neq 0$ **then**
    SUPERGRAPH(G_1)
    Store G_1 as the left subgraph of G
end if
if size(G_2) $\neq 0$ **then**
    SUPERGRAPH(G_2)
    Store G_2 as the right subgraph of G
end if
delete H
return G

When the embedding is done (cf. full version [KS16]), for defining the control bits, each node x has at most two nodes that have ingoing edges to x, one is represented as the left parent and one as the right parent of x in the edge-universal graph. The two consecutive nodes are also saved as left and right children of x. Now, when x is a switching node and we take edges (v, x) and (x, w) in the path, we save for x if parent v and child w are on the same or on the opposite side in the edge-universal graph. This defines the control bit of each universal switch in the translated universal circuit, where left and right parent and child translate to first and second input and output, respectively. We note that in order to program U_n correctly, we require that if x is the left (right) parent of v in the edge-universal graph, then v is the left (right) child of x.

Step 5. Generating the Output Circuit Description and the Programming of the Universal Circuit: After embedding the graph of the simulated circuit into the edge-universal graph U_n, we write the resulting circuit in a file using our own circuit description. In the edge-universal graph, each node stores the program bit resulting from the edge-embedding (control bit c of the corresponding universal switch in Eq. 2) and each pole stores four bits corresponding to the simulated circuit (the four control bits of the function table, c_0, c_1, c_2, c_3 in Eq. 1, their order possibly changed in Step 2). Thus, after topologically ordering U_n, one can directly write out the gate identifiers into a circuit file and the program bits to a programming file.

Our circuit description format starts with enumerating the inputs and ends with enumerating the outputs. We have universal gates denoted by U, universal switches denoted by X or Y depending on the number of outputs (X with two

outputs and Y with one). We note that we replace any gates that have only one input by wires in the UC, thus achieving our fanin-1 node optimization from Sect. 3.2. The wires are represented in the following manner:

$$\begin{array}{llllll} U & \text{in}_1 & \text{in}_2 & \text{out}_1 & & \\ X & \text{in}_1 & \text{in}_2 & \text{out}_1 & \text{out}_2 & \\ Y & \text{in}_1 & \text{in}_2 & \text{out}_1 & & \end{array} \tag{20}$$

denotes that wire out_1 (and possibly out_2) is coming from a gate with input wires in_1 and in_2. The program bits are not represented in the circuit format, but in a separate file, for each universal gate we save a four-bit number representing the control bits and for each universal switch we store the control bit. The output nodes are outputs of Y universal switches and are marked in the end of the file as $O \quad \text{o}_1 \quad \text{o}_2 \quad \ldots \quad \text{o}_v$. The circuit and its programming are given in plain text files.

Step 6. Evaluating Universal Circuits for PFE in ABY: As an example application of UCs, we implement PFE using SFE of a universal circuit. We adapted the ABY secure two-party computation framework [DSZ15] for this purpose. Firstly, since ABY uses the free-XOR optimization from [KS08a], we construct universal gates and switches with low ANDsize and ANDdepth given in Sect. 3.2. With the cost metric we consider, X and Y gates have the same AND complexity, optimized in [KS08a] and are obtained as

$$\begin{aligned} \text{out}_1 &= Y(\text{in}_1, \text{in}_2; c) = (\text{in}_1 \oplus \text{in}_2)c \oplus \text{in}_1 \\ (\text{out}_1, \text{out}_2) &= X(\text{in}_1, \text{in}_2; c) = (e \oplus \text{in}_1, e \oplus \text{in}_2) \text{ with } e = (\text{in}_1 \oplus \text{in}_2)c \end{aligned} \tag{21}$$

with ANDsize and ANDdepth of 1 for both universal switches. X gates are realized with one additional XOR gate compared to Y gates.

Our efficient implementation of generic universal gates uses Y gates yielding

$$\text{out}_1 = U(\text{in}_1, \text{in}_2; c_0, c_1, c_2, c_3) = Y[Y(c_0, c_1; \text{in}_2), Y(c_2, c_3; \text{in}_2); \text{in}_1] \tag{22}$$

with $\text{ANDsize}(U) = 3$ and $\text{ANDdepth}(U) = 2$. This universal gate implementation is generic and works in all secure computation protocols. However, for Yao's garbled circuits protocol, one can further optimize it to $\text{ANDsize}(U) = \text{ANDdepth}(U) = 1$, as in some garbling schemes such as the garbled 3-row-reduction [NPS99] the gate being evaluated remains oblivious to the evaluator.

After constructing the efficient building blocks, the output circuit file of our UC compiler is parsed, a circuit is generated accordingly and programmed with the input program bits. We conclude that our toolchain is the first implementation of Valiant's size-optimized universal circuit that supports efficient private function evaluation.

4.2 Comparison of Our PFE-Toolchain with Other PFE Protocols

Mohassel et al. in [MS13] design a generic framework for PFE and apply it to three different scenarios: to the m-party GMW protocol [GMW87], to Yao's

Table 1. The number of symmetric-key operations using different PFE protocols: Valiant's UC with SFE, the universal circuit construction from [KS08b] or Mohassel et al.'s OT-based method from [MS13]. u, v and k denote the number of inputs, outputs and gates in the simulated circuit, and k^* denotes the number of gates in the equivalent fanout-2 circuit.

Circuit	u	k	v	$k^* - k$ $(\frac{k^*}{k})$	Valiant	[KS08b]	OT-based [MS13]
`AES-non-exp`	256	31 924	128	14 539 (1.46)	$1.150 \cdot 10^7$	$2.797 \cdot 10^7$	$6.243 \cdot 10^6$
`AES-expanded`	1 536	25 765	128	11 089 (1.43)	$9.211 \cdot 10^6$	$2.206 \cdot 10^7$	$4.943 \cdot 10^6$
`DES-non-exp`	128	19 464	64	12 290 (1.63)	$7.502 \cdot 10^6$	$1.560 \cdot 10^7$	$3.639 \cdot 10^6$
`md5`	512	43 234	128	22 623 (1.52)	$1.700 \cdot 10^7$	$3.995 \cdot 10^7$	$8.681 \cdot 10^6$
`add_32`	64	187	33	58 (1.31)	35 512	55 341	19 939
`comp_32`	64	150	1	1 (1.01)	19 384	40 222	15 424
`mult_32x32`	64	6 995	64	5 079 (1.73)	$2.522 \cdot 10^6$	$4.647 \cdot 10^6$	$1.184 \cdot 10^6$
`Branching_18`	72	121	4	3 (1.02)	17 312	30 994	11 994
`CreditCheck`	25	50	1	6 (1.12)	5 056	9 348	4 198
`MobileCode`	80	64	16	0 (1.00)	12 528	13 727	5 644

garbled circuits [Yao86] and to arithmetic circuits using homomorphic encryption [CDN01]. Both the two-party version of their framework with the GMW protocol and the solution with Yao's garbled circuit protocol has two alternatives: using homomorphic encryption they achieve linear complexity $\mathcal{O}(k)$ in the circuit size k and when using a solution solely based on oblivious transfers (OTs), they obtain a construction with $\mathcal{O}(k \log k)$ symmetric-key operations. The OT-based construction in both cases is more desirable in practice, since using OT extension the number of public-key operations can be reduced significantly [IKNP03, ALSZ13].

Since the asymptotical complexity of this construction and using Valiant's UC for PFE is the same, we compare these methods for PFE. We revisit the formulas provided in [MS13] for the PFE protocol based on Yao's garbled circuits and elaborate on the number of symmetric-key operations when the different PFE protocols are used. Mohassel et al. show that the total number of switches in their framework is $4k \log_2(2k) + 1$ that are evaluated using OT extension, for which they calculate $8k \log_2(2k) + 8$ symmetric-key operations together with $5k$ operations for evaluating the universal gates with Yao's protocol. We count only the work of the party that performs most of the work, i.e., $4k$ symmetric-key operations for creating a garbled circuit with k gates and 3 symmetric-key operations (two calls to a hash function and one call to a pseudorandom function (PRF)) for each OT using today's most efficient OT extension of [ALSZ13]. Hence, according to our estimations, the protocol of [MS13] requires $12 \log_2(2k) + 4k + 12$ symmetric-key operations.

In the same way, we assume that in our case, for evaluating both the universal gates and switches, the garbler needs $4k$ symmetric-key operations. Thus, for a fair comparison, we essentially update Table 4 from the full version of [MS13, Appendix J.1], where Valiant's UC size was calculated with assumed $k^* = 2k+v$, without calculating 4 operations for the garbling.

We took our example circuit files of varying size in Table 1 from two different sources and elaborate on the resulting number of symmetric-key operations using the different constructions. The first 7 circuits we obtained from the function set of [TS15] and the last three from the FairplayPF extension of the Fairplay compiler [MNPS04, KS08b]. The example circuits that we took from [TS15] had to be converted to our desired SHDL format, which was a necessary step in order to be able to elaborate on the performance of these more complicated circuits as well. We included the INV gates in the function table of the consecutive gate and therefore, resulted in smaller gate numbers k for the equivalent SHDL circuits with arbitrary fanout. Then, these SHDL circuits were considered as input circuits for our tool.

We now compare the size of the three two-party PFE protocols: the two UC-based PFE with secure computation and the OT-based method of [MS13]. We assess our findings in Table 1. We note that our numbers are estimations, i.e., we do not consider that [MS13] works with circuits made up solely of NAND gates. Since Valiant's UC construction depends also on the number of gates with fanout more than 2 in the original circuit, we include the number of copy gates, $(k^* - k)$ in the table. We emphasize the ratio between the new number of gates k^* and the original number of gates k and conclude that in general circuits, it is well below the maximal $\frac{k^*}{k} \sim 2$. The size of the UC construction from [KS08b] obviously makes their method less efficient, in our examples using more than twice as many symmetric-key operations as the method with Valiant's UC and four times as many as Mohassel et al.'s efficient OT-based method [MS13]. We conclude that universal circuits are not the most efficient solution to perform PFE, however, we show the feasibility of generating and evaluating UCs simulating large circuits. We emphasize that even though the PFE-specific protocol from [MS13] achieves better results for PFE, universal circuits are generic and can be applied for various other scenarios (cf. Sect. 1.2), and the most efficient UC construction is Valiant's construction.

Our Experimental Results. We validated the practicality of Valiant's universal circuit construction with an efficient implementation. We ran our experiments on two Desktop PCs, each equipped with an Intel Haswell i7-4770K CPU with 3.5 GHz and 16 GB RAM, that are connected via Gigabit-LAN and give our benchmarks in Table 2. We are able to generate UCs up to around 300 000 gates of the simulated circuit, i.e., which results in billions of gates in the UC. Until now, the only implementation of universal circuits was given in [KS08b], which is outperformed by Valiant's construction already for a couple of hundred gates (cf. Figs. 4 and 5) due to its asymptotically larger complexity. We show the real practicality of UCs through experimental results proving the efficiency of our

Table 2. Running time and communication for our UC-based PFE implementation with ABY. We include the compile time, the I/O time of the UC compiler, and the evaluation time (in milliseconds) and the total communication (in Kilobytes) between the parties in GMW as well as in Yao sharing.

Circuit	UC Compile	UC I/O	GMW		Yao	
	Time (ms)	Time (ms)	Time (ms)	Communic. (KB)	Time (ms)	Communic. (KB)
AES-non-exp	2 909.2	6 331.2	5 522.08	137 561.13	2 349.35	88 417.61
AES-expanded	2 103.7	5 063.6	4 136.72	109 033.79	1 878.75	70 097.48
DES-non-exp	1 596.2	4 173.5	2 695.51	76 644.38	1 310.52	48 180.69
md5	4 043.5	8 785.4	7 041.12	169 558.83	3 547.68	110 043.59
add_32	11.4	63.8	31.97	457.77	26.49	224.77
comp_32	5.8	34.1	29.94	340.23	8.90	159.73
mult_32x32	328.9	1 443.2	1 092.46	31 053.53	539.98	18741.85
Branching_18	4.8	31.4	26.23	307.77	17.34	145.87
CreditCheck	1.2	11.4	26.25	113.35	5.67	45.15
MobileCode	3.2	26.3	25.71	202.50	28.16	103.45

implementation of PFE with the ABY framework [DSZ15]. Furthermore, due to its asymptotically smaller depth, we are also able to evaluate our generated UCs with the GMW protocol [GMW87], whereas the construction from [KS08b] was only evaluated with Yao's garbled circuit protocol. We do not directly compare our runtimes with the method of [MS13], since to the best of our knowledge, their framework has not yet been implemented.

Converting from circuit descriptions and writing into and reading out from files slows down the program significantly, but it still achieves good performance for practical circuits such as AES and DES. Our implementation in ABY can evaluate most of the circuits in both the GMW and Yao's protocols, but for some examples it runs out of memory (e.g. SHA-256). However, improvements on SFE protocols imply improvements on UC-based PFE frameworks as well. As can be seen in Table 2, the evaluation time and the communication in case of Yao's garbled cirucit protocol is about a factor of two smaller than that of the GMW protocol. This difference is due to the more efficient universal gate construction with only one gate for the case of Yao's protocol in contrast to the universal gates used in the GMW protocol with ANDsize = 3 and ANDdepth = 2.

Acknowledgements. This work has been co-funded by the European Union's 7th Framework Program (FP7/2007–2013) under grant agreement n. 609611 (PRACTICE), by the German Federal Ministry of Education and Research (BMBF) within CRISP, by the DFG as part of project E3 within the CRC 1119 CROSSING, and by the Hessian LOEWE excellence initiative within CASED. We thank Michael Zohner and Daniel Demmler for helping with the implementation in ABY, and the anonymous reviewers of Eurocrypt 2016 for their helpful comments on our paper.

References

[AF90] Abadi, M., Feigenbaum, J.: Secure circuit evaluation. J. Cryptol. **2**(1), 1–12 (1990)

[ALSZ13] Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: ACM CCS 2013, pp. 535–548. ACM (2013)

[Att14] Attrapadung, N.: Fully secure and succinct attribute based encryption for circuits from multi-linear maps. IACR Cryptology ePrint Archive 2014:772 (2014)

[BFK+09] Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.-R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 424–439. Springer, Heidelberg (2009)

[BNP08] Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP: a system for secure multi-party computation. In: ACM CCS 2008, pp. 257–266. ACM (2008)

[BPSW07] Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: ACM CCS 2007, pp. 498–507. ACM (2007)

[CCKM00] Cachin, C., Camenisch, J.L., Kilian, J., Müller, J.: One-round secure computation and secure autonomous mobile agents. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, p. 512. Springer, Heidelberg (2000)

[CDN01] Cramer, R., Damgård, I.B., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–299. Springer, Heidelberg (2001)

[CH85] Cook, S.A., Hoover, H.J.: A depth-universal circuit. SIAM J. Comput. **14**(4), 833–839 (1985)

[DDKZ13] Durnoga, K., Dziembowski, S., Kazana, T., Zajac, M.: One-time programs with limited memory. In: Lin, D., Xu, S., Yung, M. (eds.) Inscrypt 2013. LNCS, vol. 8567, pp. 377–394. Springer, Heidelberg (2013)

[DSZ15] Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: Network and Distributed System Security (NDSS 2015). The Internet Society (2015). http://encrypto.de/code/ABY

[FAL06] Frikken, K.B., Atallah, M.J., Li, J.: Attribute-based access control with hidden policies and hidden credentials. IEEE Trans. Comput. **55**(10), 1259–1270 (2006)

[FAZ05] Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving credit checking. In: ACM Electronic Commerce (EC 2005), pp. 147–154. ACM (2005)

[FGP14] Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: ACM CCS 2014, pp. 844–855. ACM (2014)

[FLA06] Frikken, K.B., Li, J., Atallah, M.J., Trust negotiation with hidden credentials, hidden policies, and policy cycles. In: Network and Distributed System Security (NDSS 2006), pp. 157–172. The Internet Society (2006)

[FVK+15] Fisch, B., Vo, B., Krell, F., Kumarasubramanian, A., Kolesnikov, V., Malkin, T., Bellovin, S.M.: Malicious-client security in Blind Seer: a scalable private DBMS. In: IEEE Symposium on Security and Privacy (S&P 2015), pp. 395–410. IEEE (2015)

[GGHZ14] Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Fully secure attribute based encryption from multilinear maps. IACR Cryptology ePrint Archive 2014:622 (2014)

[GGPR13] Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013)

[GHV10] Gentry, C., Halevi, S., Vaikuntanathan, V.: i-hop homomorphic encryption and rerandomizable Yao circuits. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 155–172. Springer, Heidelberg (2010)

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: ACM Symposium on Theory of Computing (STOC 1987), pp. 218–229. ACM (1987)

[IKNP03] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)

[KM11] Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011)

[KS08a] Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)

[KS08b] Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008)

[KS16] Kiss, Á., Schneider, T.: Valiant's universal circuit is practical. Cryptology ePrint Archive, Report 2016/093 (2016). http://eprint.iacr.org/2016/093

[LMS16] Lipmaa, H., Mohassel, P., Sadeghian, S.: Valiant's universal circuit: improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017 (2016). http://ia.cr/2016/017

[LP09a] Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009)

[LP09b] Lovász, L., Plummer, M.D.: Matching Theory. AMS Chelsea Publishing Series. American Mathematical Soc., Providence (2009)

[MNPS04] Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: USENIX Security Symposium, USENIX 2004, pp. 287–302 (2004)

[MS13] Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013)

[MSS14] Mohassel, P., Sadeghian, S., Smart, N.P.: Actively secure private function evaluation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 486–505. Springer, Heidelberg (2014)

[NPS99] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: ACM Electronic Commerce (EC 1999), pp. 129–139 (1999)

[OI05] Ostrovsky, R., Skeith III, W.E.: Private searching on streaming data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 223–240. Springer, Heidelberg (2005)

[Pin02] Pinkas, B.: Cryptographic techniques for privacy-preserving data mining. SIGKDD Explor. **4**(2), 12–19 (2002)

[PKV+14] Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S.G., George, W., Keromytis, A.D., Bellovin, S., Seer, B.: A scalable private DBMS. In: IEEE Symposium on Security and Privacy (S&P 2014), pp. 359–374. IEEE (2014)

[PSS09] Paus, A., Sadeghi, A.-R., Schneider, T.: Practical secure evaluation of semi-private functions. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 89–106. Springer, Heidelberg (2009)

[Sch08] Schneider, T.: Practical secure function evaluation. Master's thesis, University Erlangen-Nürnberg, Germany, 27 February 2008

[SS08] Sadeghi, A.-R., Schneider, T.: Generalized universal circuits for secure evaluation of private functions with application to data classification. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 336–353. Springer, Heidelberg (2009)

[SYY99] Sander, T., Young, A.L., Yung, M.: Non-interactive cryptocomputing for NC^1. In: Foundations of Computer Science (FOCS 1999), pp. 554–567. IEEE (1999)

[TS15] Tillich, S., Smart, N.: Circuits of basic functions suitable for MPC and FHE (2015). http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/

[Val76] Valiant, L.G.: Universal circuits (preliminary report). In: ACM Symposium on Theory of Computing (STOC 1976), pp. 196–203. ACM (1976)

[Weg87] Wegener, I.: The Complexity of Boolean Functions. Wiley, New York (1987)

[Yao86] Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: Foundations of Computer Science (FOCS 1986), pp. 162–167. IEEE (1986)