Efficient and Scalable Universal Circuits*

Masaud Y. Alhassan, Daniel Günther, Ágnes Kiss, Thomas Schneider
TU Darmstadt, Darmstadt, Germany
sophismay@gmail.com, guenther@rangar.de,
{kiss,schneider}@encrypto.cs.tu-darmstadt.de

April 1, 2019

Abstract

A universal circuit (UC) can be programmed to simulate any circuit up to a given size n by specifying its program inputs. It provides 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 asymptotic lower bound for the size of a UC is $\Omega(n \log n)$ and Valiant (STOC'76) provided two theoretical constructions, the so-called 2-way and 4-way UCs (i.e., recursive constructions with 2 and 4 substructures), with asymptotic sizes $5n \log_2 n$ and $4.75n \log_2 n$, respectively.

In this article, we present and extend our results published in (Kiss and Schneider, EUROCRYPT'16) and (Günther et al., ASIACRYPT'17). We validate the practicality of Valiant's UCs, by realizing the 2-way and 4-way UCs in our modular open-source implementations. We also provide an example implementation for PFE using these size-optimized UCs. We propose a 2/4-hybrid approach that combines the 2-way with the 4-way UC in order to minimize the size of the resulting UC. We realize that the bottleneck in universal circuit generation and programming becomes the memory consumption of the program since the whole structure of size $\mathcal{O}(n \log n)$ is handled by the algorithms in memory.

In this work, we overcome this by designing novel scalable algorithms for the UC generation and programming. We show that the generation, which involves topological ordering of the UC as well, can be designed to be performed block by block from top to bottom, while the programming can be performed subcircuit by subcircuit. Both algorithms use only $\mathcal{O}(n)$ memory at any point in time. We prove the practicality of our scalable design with a scalable proof-of-concept implementation for generating Valiant's 4-way UC. We note that this can be extended to work with optimized building blocks analogously. Moreover, we substantially improve the size of our UCs by including and implementing the recent optimization of Zhao et al. (ePrint 2018/943) that reduces the asymptotic size of the 4-way UC to $4.5n \log_2 n$. Furthermore, we include their optimization in the implementation of our 2/4-hybrid UC which yields the smallest UC construction known so far.

Keywords: Universal circuit, private function evaluation, function hiding, scalability

*This article is a combined and substantially extended version of [KS16] (EUROCRYPT'16) and [GKS17] (ASIACRYPT'17). We summarize the additional contributions in §1.3.

1 Introduction

Any computable Boolean function $f(x)$ can be represented as a Boolean circuit $C^g_{u,v}(x)$ with u input wires $x = (\text{in}_1, \ldots, \text{in}_u)$, v output wires $\text{out}_1, \ldots, \text{out}_v$, and g gates. Universal circuits (UCs) are programmable circuits that can simulate any Boolean function $f(x)$ up to a given size $n = u + v + g$, i.e., n is the size of the Boolean circuit description of $f(x)$. To program the UC to compute f, programming (or sometimes called control bits) are specified as further inputs $c^f = \{c_1, \ldots, c_m\}$. The UC then receives these control bits as inputs besides the input x, and computes the result as $UC(x, c^f) = f(x)$. This means that the same UC can evaluate different Boolean circuits by specifying the respective control bits. In analogy to a universal Turing machine, a universal circuit allows to turn any function into data in the form of a program description. Thus, the size-depth problem of UCs is related to the time-space problem of Turing machines [Val76].

Several efficient constructions considering both the size and the depth of UCs were proposed. Valiant proposed in [Val76] an asymptotically size-optimal UC construction with size $\Theta(n \log n)$ and depth $\mathcal{O}(n)$ [Weg87]. He provides two constructions, based on so-called edge-universal graphs (EUGs) that utilize either 2 or 4 subcircuits, also called 2-way or 4-way UCs, respectively. The asymptotic complexity of the 4-way UC is $\sim 4.75 n \log_2 n$ which is smaller than that of the 2-way UC of $\sim 5 n \log_2 n$ [Val76]. The 4-way UC has been further improved in [ZYZL18], where its size is reduced to $\sim 4.5 n \log_2 n$. An asymptotically depth-optimal construction with depth $\Theta(d)$ that simulates circuits with depth d was proposed in [CH85], but it has a significantly larger size of $\mathcal{O}(n^3 d / \log n)$. In our paper, due to the applications in cryptography that we revisit in §1.1, we concentrate on the existing size-optimized UCs, especially that proposed by Valiant [Val76] with size $\Theta(n \log n)$ with the optimization presented by Zhao et al. in [ZYZL18].

1.1 Applications of Universal Circuits

Size-optimized universal circuits have many applications, which we review here and refer to the original publications for a more detailed description.

Private Function Evaluation (PFE)

The most prominent application of universal circuits is the secure evaluation of private functions based on *secure function evaluation* (SFE) or *secure computation*. SFE enables two parties P_1 and P_2 to evaluate a publicly known function $f(x, y)$ on their respective private inputs x and y, ensuring that none of the participants learns anything about the other participant's input besides the output of the computation. Many secure computation protocols use Boolean circuits for representing the desired functionality, such as Yao's garbled circuit protocol [Yao82, Yao86, LP09a] and the GMW protocol [GMW87]. In some applications the function itself should be kept private. 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 x. P_2 should learn no information about f besides an upper bound on the size of the circuit describing the function, and P_1 should learn nothing about x beyond what can be inferred from the result $f(x)$.

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. For this, P_1 provides the control bits c^f for the UC and P_2 provides his private input x into an SFE protocol that computes $UC(x, c^f)$. Here, the UC is a public function and the control bits c^f – and therefore the function f – and input x are

kept private due to the properties of SFE. The first implementation of PFE was provided in [KS08b, Sch08], which extends the Fairplay secure computation framework [MNPS04] with universal circuits. This construction achieves a non-optimal asymptotic size of $\mathcal{O}(n \log^2 n)$ and depth $\mathcal{O}(n \log n)$. We have shown in [KS16] that it results in larger UCs than Valiant's constructions for all reasonable circuit sizes in practice. The complexity of PFE in this case is determined mainly by the size and depth of the UC, while 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. UC-based PFE can be easily integrated into any SFE framework and can directly benefit from recent optimizations. For instance, *outsourcing UC-based PFE* to two or multiple servers using XOR secret sharing is directly possible with outsourced SFE [KR11]. The non-interactive secure computation protocol of [AMPR14] can be generalized to obtain a *non-interactive PFE* protocol [LMS16]. Moreover, with UC-based PFE, evaluating public and private parts of a functionality can easily be performed together without modifying the underlying secure computation framework.

In [KM11], Katz and Malka presented an alternative approach for PFE that does not rely on UCs. They use additively homomorphic public-key encryption as well as a symmetric-key encryption scheme and achieve constant-round PFE with linear $\mathcal{O}(n)$ 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. Their protocol is secure against semi-honest adversaries, uses Yao's garbled circuits [Yao86], and has recently been improved in [BBK18], where the authors modify the algorithm to perform one full execution from which information can be reused in subsequent more efficient executions of the protocol. Mohassel and Sadeghian consider PFE with semi-honest adversaries in [MS13] and propose a generic PFE framework that can be instantiated with different secure computation protocols. Their first protocol uses homomorphic encryption with which they achieve linear complexity $\mathcal{O}(n)$ in the circuit size n and their second protocol relies solely on oblivious transfers (OT), that results in a method with $\mathcal{O}(n \log n)$ symmetric-key operations. The OT-based construction from [MS13] or PFE using UCs are more desirable than the linear homomorphic encryption-based methods in practice, since using OT extension, the number of expensive public-key operations can significantly be reduced, such that it is independent of the number of OTs [IKNP03, ALSZ13]. Biçer et al. [BBKL17] improve the communication of the OT-based PFE protocol of [MS13] by around 40%. The asymptotic 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 §8. Mohassel et al. extend the framework from [MS13] to malicious adversaries in [MSS14] with linear complexity $\mathcal{O}(n)$, using additively homomorphic encryption. Active security of UC-based PFE is achieved by using a secure computation protocol with active security. Even though their claimed better efficiency, to the best of our knowledge, these protocols have not yet been implemented, and they are not as generally applicable as PFE with UCs, e.g., they cannot be easily combined with secure evaluation of public functions.

Semi-private function evaluation (semi-PFE) has been proposed in [PSS09], and allows for PFE where the function f is in a set of functions $\mathcal{F}$ known by both parties. This relaxes the necessary topology hiding requirement of generic PFE. Yao's garbled circuit can be used for evaluating circuits of the same topology as shown in [PKV+14]. Recently, an automated approach for semi-PFE has been proposed in [KKW17], where $f \in \mathcal{F}$ have varying topologies, for which a container topology is found that can be programmed to compute any of the available topologies. This approach has been

further improved in [Kol18], where a modified garbled circuit protocol allows for efficient semi-PFE with linear communication in the size of the largest circuit in $\mathcal{F}$. However, semi-PFE does not suffice for generic PFE where we have an exponential number of possible circuit topologies.

Applications of PFE

PFE can be applied in scenarios where one of the parties wants to keep the evaluated function private. One of the first applications for PFE was *privacy-preserving checking for credit worthiness* [FAZ05], where not only the loanee's data, but also the loaner's function that computes if the loanee is eligible for a credit needs to be kept private. The original scheme, using garbled circuits, can represent simple policies, but by evaluating a UC their scheme can be extended to more complicated credit checking policies. [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. [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. PFE allows for running *proprietary software* on private data, such as privacy-preserving evaluation of *diagnostic programs* that 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 diagnostics [BFK+09] and remote software fault diagnosis, where 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. In [PKV+14, FVK+15], universal circuits are used for hiding *queries in private database management systems* (DBMSs). The Blind Seer DBMS [FVK+15] 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. Further applications of PFE given in [MS13] are *evaluation of branching programs on encrypted data* [IP07] and *privacy-preserving intrusion detection* [NSMS14].

UC Applications Beyond PFE

Besides being used for PFE, UCs can be applied in various other scenarios. Efficient *verifiable 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 UCs for reducing the verifier's preprocessing step. In [GHV10], a DDH-based *multi-hop homomorphic encryption* scheme is proposed that uses re-randomizable garbled circuits, for which UCs are used to achieve function privacy. 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. As described in [Att14], the *Attribute-Based Encryption* (ABE) schemes [GGH+13b] [GVW13] for any polynomial-size circuits can be turned into ciphertext-policy ABE by using UCs. The ABE scheme of [GGHZ14] also uses UCs. Universal circuits can be

applied for program obfuscation. Candidates for *indistinguishability obfuscation* are constructed using a UC as a building block in [GGH+13a, BV15]. The algorithm of [GGH+13a] has been implemented in [BOKP15], which can be improved using Valiant's UC implementation [KS16]. *Direct program obfuscation* was proposed in [Zim15], where the circuit is a secret key to a UC. [LMS16] mentions that UCs can be applied for secure two-party computation in the batch execution setting, where the cost of evaluating Yao's garbled circuits is amortized if the same circuit – a UC – is evaluated [HKK+14, LR15]. This protocol has been made round-optimal in [MR17].

Implied Theoretical Results

We mention two theoretical results relying on UCs. Both the depth-optimized UC from [CH85] and Valiant's size-optimized UCs were adapted in [BFGH10] to construct *universal quantum circuits*. The design of *universal parallel computers* was inspired by Valiant's UCs as well [GP81, Mey83].

1.2 Our Contributions and Outline

In §2, we recapitulate the necessary preliminaries for our work. We revisit the asymptotically size-optimal UCs of [Val76] in §3. This complex construction makes use of an internal graph representation and programs a so-called edge-universal graph (§3.1). Thereafter, we describe how an edge-universal graph can be translated into a universal circuit (§3.2). Finally, we revisit the 2-way (§3.3) and 4-way UC (§3.4) and the improved building block proposed by Zhao et al. [ZYZL18] for the latter.

Our modular programming algorithm (§4). We detail our modular algorithm for programming a universal circuit that provides the description of the input function f as program bits c^f to the UC, both for Valiant's 2-way and 4-way UCs. Our method consists of two steps, the block edge-embedding (§4.1) and the recursion point edge-embedding (§4.2).

New universal circuit constructions and extensions (§5). We describe Lipmaa et al.'s generalization of Valiant's UC to any k-way UC (§5.1), and detail how our modular programming algorithm from §4 can be directly generalized for this extension. We continue with a new 3-way UC (§5.2) that is predicted to be more efficient than the existing UCs. However, after providing modular building blocks for this UC, we show that it is asymptotically larger than Valiant's UCs, due to an optimization that cannot be applied for one of its building blocks. Then, we propose a hybrid UC construction (§5.3) that can efficiently combine k-way UCs for multiple values of k. With this, we combine Valiant's 2-way and 4-way UCs to achieve the smallest universal circuit known so far. Lastly, we provide our scalable UC algorithms (§5.4), which allow for generating and programming UCs with only linear $\mathcal{O}(n)$ memory instead of handling the whole structure of size $\mathcal{O}(n \log n)$ in memory at once.

Optimized size and depth of UCs (§6). We compare the asymptotic (§6.1) and concrete (§6.2) sizes of Valiant's (2-way and 4-way) UCs and that of different k-way UCs. We show that of all k-way UCs of Lipmaa et al. [LMS16], Valiant's 4-way UC provides the best results for large circuits. We include size optimizations, achieving a linear concrete improvement for all UCs. Moreover, we show that our 2/4 hybrid method for generating UCs improves over the 4-way UCs, i.e., both when using Valiant's 4-way UC and the optimized 4-way UC of [ZYZL18].

Implementation of Valiant's UCs and experiments (§7). We detail the steps of our algorithm for a practical realization of Valiant's UC construction, and implement the 2-way and recently optimized 4-way UCs as well as our 2/4 hybrid UC construction. We note that our implementation is the

first implementation that includes the optimization of Zhao et al. [ZYZL18], which achieves the best size $\sim 4.5n \log_2 n$. We describe the architecture of our UC compiler (§7.1). We experimentally evaluate the performance of our UC generation and programming algorithms with a set of example circuits (§7.2). We provide the evaluation of our scalable 4-way UC as well, and compare it with our memory-based implementation of Valiant's 4-way UC.

Toolchain for private function evaluation using universal circuits (§8). We provide the implementation of an example application for universal circuits, namely of private function evaluation (PFE) by extending the ABY secure function evaluation framework [DSZ15] to evaluate our universal circuits (§8.1). We provide the first implementation for PFE with $\mathcal{O}(n \log n)$ complexity and show experimental results for performing PFE (§8.2). We theoretically compare PFE with UCs with other state-of-the-art approaches for PFE (§8.3).

1.3 Additions to Conference Versions

This journal article is a significantly extended and improved version of the conference publications [KS16] and [GKS17]. Our added contributions are as follows.

1. *Optimizations.* We included the optimized building block of [ZYZL18] in our 4-way and hybrid implementations as well as in the size and depth comparisons. This allows us to compare all state-of-the-art methods for UCs. This is the first implementation of their construction, which has the lowest asymptotic and concrete sizes known so far.

2. *Scalability.* We extend our design and implementation with a scalable 4-way UC construction based on Valiant's 4-way UC, which reduces the memory complexity from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$ when generating and programming the universal circuit. This construction involves a novel layer-by-layer approach for generating and topologically ordering the universal circuit and programs the structure according to the recursion steps, i.e., subcircuit by subcircuit.

3. *Universal circuit depths.* We examine the depth of the universal circuits in addition to their sizes, since though being optimized for the latter, some applications also require to minimize the former, e.g., PFE via secure function evaluation with the GMW protocol [GMW87] which in contrast to Yao's garbled circuits allows to precompute all symmetric cryptographic operations [SZ13], but its number of communication rounds depends on the depth of the universal circuit.

4. *Comparison and implementation.* In our previous works, we have compared the 2-way and 4-way UCs with each other and with the only other existing UC of [KS08b]. In this work, we implement the hybrid method that uses both 2-way and 4-way UCs and achieves the best concrete size for all circuit sizes. We also implement our new scalable 4-way UC construction, which utilizes very different algorithms than those applied before for UC generation. We compare these methods with respect to runtime, communication and memory consumption.

2 Preliminaries

As preliminaries for our paper we introduce the graph and circuit theoretic background in §2.1 and §2.2, respectively. We provide a summary of all our notations and abbreviations in Appendix A.

2.1 Graph Theory

In this section, we describe the graph theoretic preliminaries necessary for our work.

Definition 1. *The number of incoming [outgoing] edges of a node is called its* indegree [outdegree]. *A graph has* fanin [fanout] ρ *if the indegree [outdegree] of all its nodes is at most* ρ.

We denote by $\Gamma_\rho(n)$ the set of all directed acyclic graphs with n nodes and fanin and fanout ρ.

Definition 2. *Let* $G = (V, E)$ *be a* directed graph *with set of* nodes $V = \{1, \ldots, n\}$ *and* edges $E \subseteq V \times V$. *A mapping* $\eta^G : V \to \{1, \ldots, n\}$ *is called* topological order *if* $(i, j) \in E$ *implies that* $\eta^G(i) < \eta^G(j)$ *and* $\forall i, j \in V : \eta^G(i) = \eta^G(j)$ *means that* $i = j$. *In short,* $i > j$ *implies that there is no edge or directed path from* i *to* j.

A topological order of $G \in \Gamma_\rho(n)$ can be found with computational complexity $\mathcal{O}(\rho n)$. Further on, we require a labelling of the nodes in a topological order.

Definition 3. Edge-embedding *is a mapping from graph* $G = (V, E)$ *into* $G' = (V', E')$ *that maps* V *into* V' *one-to-one, with possible additional nodes in* V', *i.e.,* $V \subseteq V'$ *and* E *into directed paths in* E', *such that all paths are pairwise edge-disjoint, i.e., an edge can be used only in one path.*

Theorem 1 (Kőnig-Hall theorem)**.** *Given a directed acyclic graph (DAG)* $G \in \Gamma_2(n)$, *the set of edges* E *can be separated into two disjoint 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 [Kő31, Val76, LP09b].*

Proof of Theorem 1. Given the set of nodes in topological order $V = \{1, \ldots, n\}$, 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 $(i, j) \in E$. We can easily see that the fanin and fanout of the resulting bipartite graph is also 2. 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 of at most 2, such coloring can be found directly with the following method:

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**

This coloring can be performed in $\mathcal{O}(n)$ steps and it defines the edges in E_1 and E_2, such that E_1 contains the edges colored with color one and E_2 the ones with color two and $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. □

The Kőnig-Hall theorem was used in [KS16, LMS16] to provide a 2-coloring algorithm for a graph with fanin and fanout 2. In its originally proposed form, however, Kőnig's theorem [Kő31, LP09b] applies also for k-coloring any graph with at most k incoming and outgoing edges for each of its

nodes. This transformation can be easily generalized to graphs in $\Gamma_k(n)$, in which case the resulting bipartite graph will have fanin and fanout k. We review this theorem and the corresponding algorithm here.

Theorem 2 (Kőnig's theorem)**.** *If $\overline{G}$ is bipartite and its nodes have at most k incoming and outgoing edges, then the number of colors necessary to color $\overline{G}$ is k.*

Proof of Theorem 2. Take colors $\{1, \ldots, k\}$, and greedily color edges. Let us assume that at some point the coloring stops because we cannot color more edges. In this step, (w_i, z_j) is an uncolored edge. If we look at the colors of the neighbours of w_i and z_j, we can define the set of available colors for both nodes. There is at least one color both for w_i and z_j due to the fanin and fanout restriction, but there is no color which is available for both nodes, otherwise we could color (w_i, z_j).

There is a color that is used in an edge incident to w_i, e.g., color a, but not on an edge incident to z_j. In the same way, we can find another color b, that is used in an edge incident to z_j, but not to w_i. Take the longest unique path P from w_i that uses colors a and b alternatingly.

Indirectly, assume that this path also contains z_j. It then terminates in z_j due to the fact that z_j is not incident with an edge colored with a. Then, $P \cup (w_i, z_j)$ is an odd cycle, which is impossible since $\overline{G}$ is bipartite. Therefore, p does not contain z_j, and we can exchange colors a and b on path P and color (w_i, z_j) with color a.

This process is continued until there are no uncolored edges in $\overline{G}$. □

2.2 Circuit Theory

Definition 4. *The* fanin [fanout] *of a circuit can be defined analogously to the fanin [fanout] of a graph, i.e., based on the maximal number of incoming [outgoing] wires of all its gates, inputs and outputs.*

Theorem 3. *A circuit $C^{\hat{g}}_{u,v}$ with u inputs, $\hat{g}$ gates and v outputs and fanin and fanout $\rho > 2$ can be transformed to a circuit $C^{g}_{u,v}$ with fanin and fanout 2.*

Proof of Theorem 3. Shannon's expansion theorem [Sha49, Sch08] describes how gates with larger fanin can be reduced to gates with two inputs by adding additional gates, which results in a circuit $C^{\tilde{g}}_{u,v}$ with $\tilde{g}$ fanin 2 gates. It was proven in [Val76] that the general case, where the fanout of the circuit can be any integer $\rho \geq 2$, can be transformed to the special case when $\rho \leq 2$ by introducing copy gates, each of which eliminates one from the extra fanout of the original gate. We place 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]. Thus, the class of Boolean functions with u inputs and v outputs that can be realized by acyclic circuits with $\tilde{g}$ gates and arbitrary fanout, can also be realized with an acyclic fanout-2 circuit with $\tilde{g} \leq g \leq 2\tilde{g} + v$ gates. □

Definition 5. *We can regard $C^{g}_{u,v}$ with u inputs, v outputs and g gates as a $\Gamma_2(n)$ graph G – which we commonly refer to as the* graph of circuit *$C^{g}_{u,v}$ – with $n = u + v + g$ by creating a node for each input, gate and output, and an edge for each wire in $C^{g}_{u,v}$.*

3 Valiant's Universal Circuit Constructions

In any circuit $C^{\hat{g}}_{u,v}$, the inputs of the $\hat{g}$ gates are either connected to one of the u inputs, to the output of another gate, or are assigned a fixed constant. Due to the nature of Valiant's edge-universal graph construction, the input circuit must have fanin and fanout 2, which can be achieved with the transformations described in §2.2 and implemented in [KS08b, KS16]. From here on, we assume that our input circuit $C^{g}_{u,v}$ has u inputs, g gates and v outputs and fanin and fanout 2.

The size of a function f represented by a circuit $C^{g}_{u,v}$ with fanin and fanout 2 is $n = u + v + g$, which can be represented as a graph $G \in \Gamma_2(n)$. In this section, we describe Valiant's universal circuit constructions [Val76, Weg87] that can be programmed to evaluate any function of size n. We explain the general idea behind Valiant's construction [Val76] in §3.1 and §3.2, and the 2-way and 4-way UCs along with improvements of [KS16, LMS16, GKS17, ZYZL18] in §3.3 and §3.4, respectively.

3.1 Valiant's Edge-Universal Graph Construction

Valiant's UC construction relies on the notion of so-called edge-universal graphs that are then translated to universal circuits [Val76].

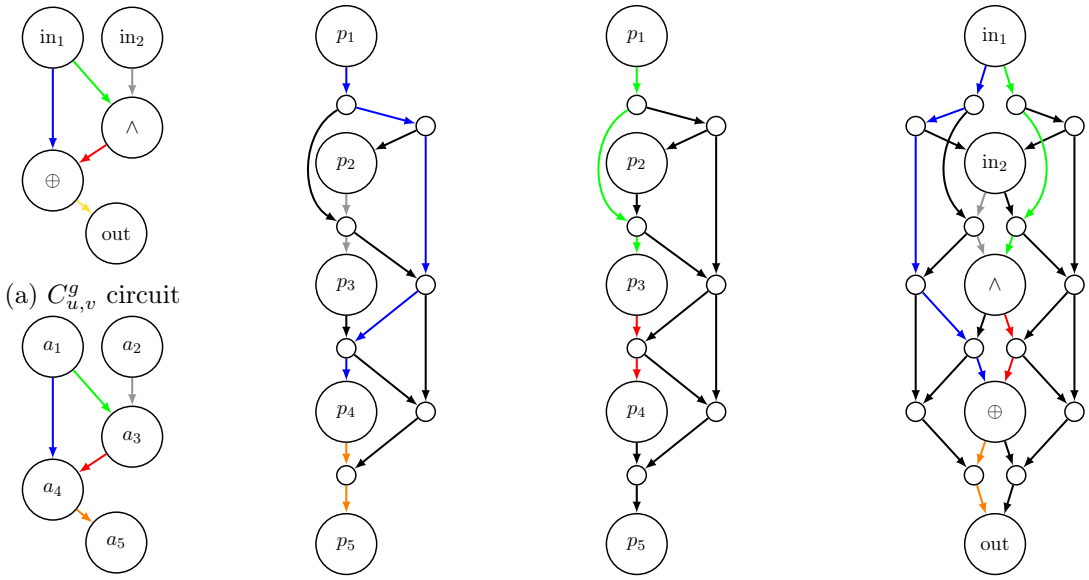
Definition 6. *A graph $U_n(\Gamma_\rho) = (V_U, E_U)$ is an* edge-universal graph *(EUG) for $\Gamma_\rho(n)$ if every graph $G = (V, E)$ in $\Gamma_\rho(n)$ can be edge-embedded (cf. Def. 3) into $U_n(\Gamma_\rho)$.*

An EUG $U_n(\Gamma_\rho)$ has distinguished nodes called *poles* $P = \{p_1, \ldots, p_n\} \subseteq V_U$ where each node $a \in V = \{1, \ldots, n\}$ is mapped to exactly one pole with an injective mapping $\varphi^V : V \to V_U$. This mapping is defined by a concrete topological order η^G of the original graph G with $\varphi^V(a) = p_{\eta^G(a)}$, i.e., every node in G has a corresponding pole in $U_n(\Gamma_\rho)$. Besides the poles, $U_n(\Gamma_\rho)$ might have additional nodes that enable the edge-embedding (cf. §2.1). For each edge $(a_i, a_j) \in E$ we then define a path of variable length z between the corresponding poles $\varphi^V(a_i) = p_{\eta^G(a_i)} = b_1$ and $\varphi^V(a_j) = p_{\eta^G(a_j)} = b_z$ as $(b_1, \ldots, b_z)$, where $b_1, \ldots, b_z \in V_U$. All these paths are edge-disjoint, i.e., they do not use any edge in $U_n(\Gamma_\rho)$ in more than one path (cf. §2.1).

Let $U_n(\Gamma_1)$ be an EUG for graphs in $\Gamma_1(n)$ with poles $P = \{p_1, \ldots, p_n\}$. The poles have fanin and fanout 1, while all other nodes have fanin and fanout 2. An EUG $U_n(\Gamma_\rho)$ for $\rho \geq 2$ can be created by taking ρ instances of $U_n(\Gamma_1)$ EUGs, and merging each pole p_i with its multiple instances, allowing the poles to have fanin and fanout ρ. Let $U_n(\Gamma_\rho) = (V'_U, E'_U)$ be an EUG with fanin and fanout ρ, constructed with $U_n(\Gamma_1)_1 = (V_1, E_1), \ldots, U_n(\Gamma_1)_\rho = (V_\rho, E_\rho)$. P contains the merged poles and $V'_U = P \cup_{i=1}^{\rho} V_i \backslash P_i$ and $E'_U = \cup_{i=1}^{\rho} E_i$. Now, every pole in $U_n(\Gamma_\rho)$ has at most ρ, and every node has at most two inputs and outputs.

We give an example for better understanding. Let $G = (V, E)$ be the graph with 5 nodes shown in Fig. 1b, which is the graph of the circuit in Fig. 1a. Our aim is to edge-embed G into EUG $U_5(\Gamma_2)$. Therefore, we use two instances of $U_5(\Gamma_1)$: $U_5(\Gamma_1)_1$ in Fig. 1c and $U_5(\Gamma_1)_2$ in Fig. 1d. The edges (a_1, a_4), (a_2, a_3) and (a_4, a_5) are embedded in $U_5(\Gamma_1)_1$, and the edges (a_1, a_3) and (a_3, a_4) in $U_5(\Gamma_1)_2$. Merging the poles of $U_5(\Gamma_1)_1$ and $U_5(\Gamma_1)_2$ produces $U_5(\Gamma_2)$ shown in Fig. 1e.

Recursion Base. Valiant's construction is recursive, and the recursion base graphs for up to 6 nodes are shown in [Val76, Fig. 3] and [KS16, Fig. 1]. $U_1(\Gamma_1)$ is a single pole, $U_2(\Gamma_1)$ and $U_3(\Gamma_1)$ are two and three connected poles, respectively. Valiant provides hand-optimized EUGs for $U_4(\Gamma_1)$, $U_5(\Gamma_1)$ and $U_6(\Gamma_1)$, with 3, 7 and 9 additional nodes, respectively (cf. [Val76, Fig. 3]).



(a) $C^g_{u,v}$ circuit

(b) $\Gamma_2(5)$ graph G

(c) $U_5(\Gamma_1)_1$

(d) $U_5(\Gamma_1)_2$

(e) EUG $U_5(\Gamma_2)$

Figure 1: Fig. 1a shows an example circuit and Fig. 1b the corresponding $\Gamma_2(5)$ graph G. Figs. 1c-1d show the edge-embedding of G into two $U_5(\Gamma_1)$ instances with poles $(p_1, \ldots, p_5)$. Fig. 1e shows the edge-embedding of G into the $U_5(\Gamma_2)$ graph of the universal circuit.

3.2 Translating Edge-Universal Graphs into Universal Circuits

In this section, we define universal circuits (UCs) and describe how an edge-universal graph is translated into a universal circuit.

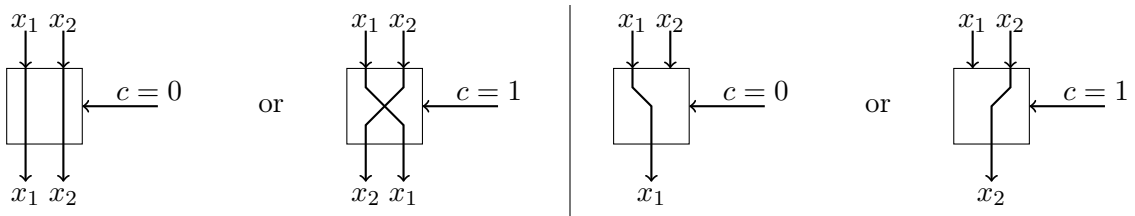
Definition 7. *A* universal circuit *UC is a Boolean circuit of size $\mathcal{O}(n \log n)$ that can be programmed to compute any circuit $C^g_{u,v}$ up to a given size n by defining a set of programming bits c^f such that $UC(x, c^f) = C^g_{u,v}(x)$.*

Every node $w \in V_U$ fulfills a task when $U_n(\Gamma_2)$ is translated to a universal circuit. Programming the UC means specifying its control bits along the paths defined by the edge-embedding and by the gates of circuit $C^g_{u,v}$. Depending on the number of incoming and outgoing edges and its type, a node is translated as described below.

G1 If w is a pole and corresponds to an input (one of the first u poles) or an output (one of the last v poles) in G, then w is an *input or output* in $C^g_{u,v}$ as well.

G2 If w is no pole and has indegree 1 and outdegree 2, this node has been placed to copy its input to its two outputs. Therefore, when translated to a UC, w is replaced by multiple outgoing wires in the parent node (as described in [KS16]), since the UC does not need to fulfill the fanout 2 restriction. In $U_n(\Gamma_2)$, w is added due to the fanout 2 restriction in the EUG necessary for the edge-embedding.

G3 If w is no pole and has indegree and outdegree 1, w is removed and replaced by a wire between its parent and child nodes.



(a) X-Switching Block

(b) Y-Switching Block

Figure 2: Programmable switching blocks.

G4 If w is a pole and corresponds to a gate (poles $\{u+1, \ldots, u+g\}$) in G, w is programmed as a *universal gate* (UG). A 2-input UG supports any of the 16 possible gate types represented by 4 control bits of the gate table (c_1, c_2, c_3, c_4). It implements function U: $\{0,1\}^2 \times \{0,1\}^4 \to \{0,1\}$ that computes

$$U(x_1, x_2, c_1, c_2, c_3, c_4) = \overline{x_1}\ \overline{x_2} c_1 + \overline{x_1} x_2 c_2 + x_1 \overline{x_2} c_3 + x_1 x_2 c_4. \tag{1}$$

G5 If w is no pole and has indegree and outdegree 2, w is programmed as an *X-switching block*, that computes $X : \{0,1\}^2 \times \{0,1\} \to \{0,1\}^2$ with $X((x_1, x_2), c) = (x_{1+c}, x_{2-c})$ as shown in Fig. 2a. The inputs of an X-switching block are forwarded to its outputs, switched or not switched, depending on control bit c.

G6 If w is no pole and has indegree 2 and outdegree 1, w is programmed as a *Y-switching block* that computes $Y : \{0,1\}^2 \times \{0,1\} \to \{0,1\}$ with $Y((x_1, x_2), c) = x_{1+c}$ as visualized in Fig. 2b. The inputs of a Y-switching block are forwarded to its output depending on the control bit c, i.e., it provides the functionality of a 2-input multiplexer.

We note that the u inputs and the v outputs can be ordered arbitrarily within themselves as long as the inputs are kept before the g 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. Note that optionally it is possible to modify the input circuit such that the outputs of the last v gates in order are the outputs of the circuit [KM11], which can be achieved by inserting at most v copy gates.

The nodes programmed as UG (**G4**), X-switching block (**G5**) or Y-switching block (**G6**) are so-called *programmable blocks*. This means that a control bit c or vector $\overline{c} = (c_1, c_2, c_3, c_4)$ is necessary besides the two inputs to define their behavior. The universal gates are programmed according to the simulated gates in $C^g_{u,v}$ and the universal switches according to the paths defined by the edge-embedding of the graph of the circuit G into the edge-universal graph $U_n(\Gamma_2)$. 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 is programmed accordingly. In §7.1, we describe efficient implementations of programmable blocks. The control bits and vectors build up the programming of the UC, i.e., c^f.

3.3 Valiant's 2-Way UC Construction

We described in §3.1 that a $U_n(\Gamma_\rho)$ EUG can be constructed of ρ instances of $U_n(\Gamma_1)$ EUGs. Valiant [Val76] provides an EUG for $\Gamma_1(n)$ graphs, two of which can build an EUG for $\Gamma_2(n)$ graphs, which suffices for circuits with 2-input gates. Let $P = \{p_1, \ldots, p_n\}$ be the set of poles in $U_n(\Gamma_1)$ that have indegree and outdegree 1, corresponding to the inputs, gates and outputs of the input circuit $C^g_{u,v}$, i.e., poles $P_{\text{in}} = \{p_1, \ldots, p_u\}$ correspond to the inputs, $P_{\text{gate}} = \{p_{(u+1)}, \ldots, p_{(u+g)}\}$ to the gates, $P_{\text{out}} = \{p_{(u+g+1)}, \ldots, p_n\}$ to the outputs. Valiant's 2-way EUG for $\Gamma_1(n)$ graphs $U^{(2)}_n(\Gamma_1)$ of size $\sim 2.5n \log_2 n$ is shown in Fig. 3. The corresponding UC has twice the size $\sim 5n \log_2 n$, since it corresponds to the EUG for $\Gamma_2(n)$ graphs.

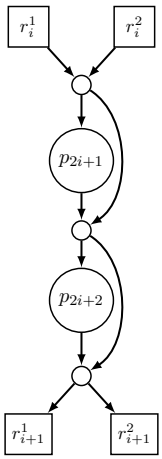


Figure 3: Body block $B^{(2)}$ of Valiant's 2-way EUG $U^{(2)}_n(\Gamma_1)$ [Val76].

The recursive construction shown in Fig. 3 works as follows: the rectangles are special nodes that build up the set of poles in the next recursion step, i.e., $R^1_{\lceil \frac{n}{2}-1 \rceil} = \{r^1_1, \ldots, r^1_{\lceil \frac{n}{2}-1 \rceil}\}$ and $R^2_{\lceil \frac{n}{2}-1 \rceil} = \{r^2_1, \ldots r^2_{\lceil \frac{n}{2}-1 \rceil}\}$ that are the poles of two smaller edge-universal graphs called subgraphs. EUGs are built with these poles which produces new subgraphs with size $\lceil \frac{\lceil \frac{n}{2}-1 \rceil}{2} - 1 \rceil$, s.t. we have four subgraphs at the next level, etc.

Valiant offers the main, so-called *body block* $B^{(2)}$ (Fig. 3) consisting of 2 poles (large circles), 3 nodes (small circles) as well as 4 recursion points (squares) [Val76]. We note that the top [bottom] block does not need the upper [lower] recursion points since its poles are the inputs [outputs] in the block. Therefore, we presented optimized so-called head $H^{(2)}$ and tail $T^{(2)}$ blocks that occur in the top and bottom of a skeleton, respectively, in [GKS17, Figs. 2b-2e].

This construction is called the *2-way EUG or UC construction* since there are two sets of recursion nodes at each recursion step. We provided an open-source implementation of this 2-way UC optimized for PFE in [KS16]. In concurrent and independent related work, Lipmaa et al. also show the practicality of Valiant's 2-way UC [LMS16]. They decrease its total number of gates compared to that of Valiant's block (cf. Fig. 3) by one. However, the number of AND gates is exactly the same and therefore their improvement does not affect PFE using UCs, where XOR gates are evaluated for free [KS08b].

Theorem 4 ([Val76]). *The resulting 2-way EUG is edge-universal and therefore the resulting circuit is universal.*

Proof of Theorem 4 [Val76]. We recapitulate the proof from [Val76] that $U^{(2)}_n(\Gamma_1)$ is edge-universal for $\Gamma_1(n)$, such that any graph with n nodes and fanin and fanout 1 can be edge-embedded into $U^{(2)}_n(\Gamma_1)$. According to the definition of edge-embedding, it has to be shown that given any $\Gamma_1(n)$ graph $G = (V, E)$, for any $(i, j) \in E$ and $(k, l) \in E$ we can find pairwise edge-disjoint paths from p_i to p_j and from p_k to p_l in $U^{(2)}_n(\Gamma_1)$. As before, the labelling of nodes $V = \{1, \ldots, n\}$ in G is according to a topological order of the nodes.

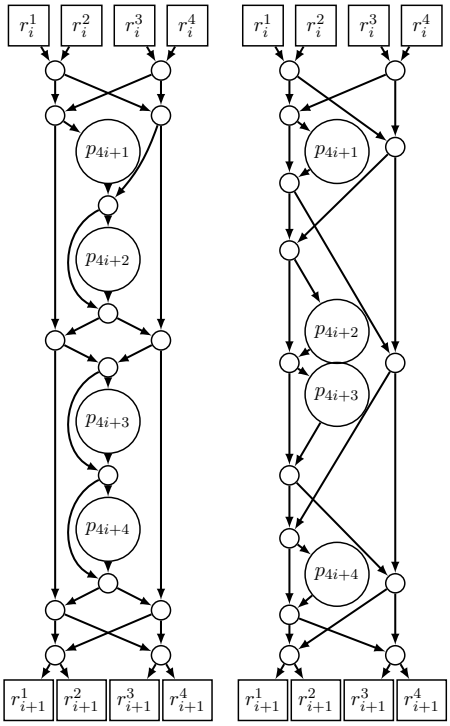
Firstly, each two neighbouring poles of the edge-universal graph, p_{2s} and p_{2s+1} for $s \in \{1, \ldots, \lceil \frac{n}{2} \rceil\}$, are thought of as merged superpoles, with their fanin and fanout becoming 2. In a similar manner, any $G \in \Gamma_1(n)$ graph can be regarded as a $\Gamma_2(\lceil \frac{n}{2} \rceil)$ 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 disjoint 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 in Theorem 1. The edges in E_1 are embedded as directed paths in $R^1_{\lceil \frac{n}{2}-1 \rceil}$, and the edges in E_2 as directed paths in $R^2_{\lceil \frac{n}{2}-1 \rceil}$. 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 $U^{(2)}_n(\Gamma_1)$ as well. Thus, the edge coming into a superpole (p_{2s}, p_{2s+1}) in E_1 is embedded as a path through r^1_{s-1}, while the edge going out of the pole in E_1 is embedded as a path through r^1_s in the appropriate subgraph. Similarly, the edges in E_2 are simulated as edges through r^2_{s-1} and r^2_s. These paths can be chosen disjoint according to the induction hypothesis. Finally, the paths from r^1_{s-1} and r^2_{s-1} to superpole (p_{2s-1}, p_{2s}) as well as the paths from (p_{2s-1}, p_{2s}) to r^1_s and r^2_s can be chosen edge-disjoint due to the skeleton shown in Fig. 3. 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]. With the building blocks described in §3.2, it is easy to see that the resulting Boolean circuit is universal. □

3.4 Valiant's 4-Way UC Construction

Similarly to the 2-way EUG construction (cf. §3.3), Valiant provides a more efficient *4-way EUG or UC construction* [Val76] for $\Gamma_1(n)$ graphs which can be extended to an EUG for $\Gamma_2(n)$ graphs by utilizing two instances $U^{(4)}_n(\Gamma_1)_1$ and $U^{(4)}_n(\Gamma_1)_2$ as described in §3.1. $U^{(4)}_n(\Gamma_1)$ has a 4-way recursive structure, i.e., at each recursion step, nodes in special sets $R^1_{\lceil \frac{n}{4}-1 \rceil}$, $R^2_{\lceil \frac{n}{4}-1 \rceil}$, $R^3_{\lceil \frac{n}{4}-1 \rceil}$ and $R^4_{\lceil \frac{n}{4}-1 \rceil}$[2] are the poles in the next recursion step (cf. Fig. 4a). The recursion base is the same as for the 2-way UC construction described in §3.1. This construction results in UCs of smaller size $\sim 4.75 n \log_2 n$ but has a more complicated structure and programming algorithm. We have studied and implemented this universal circuit in [GKS17], and recapitulate our results here and in §7. Valiant offers the main, so-called *body block* $B^{(4)}$ (Fig. 4a) consisting of 4 poles (large circles), 15 nodes (small circles) as well as 8 recursion points (squares). As before, we provide so-called *head* $H^{(4)}$ and *tail* $T^{(4)}$ *blocks* that occur at the top and bottom of a skeleton in [GKS17, Figs. 4b-4i], respectively.

Recently, Zhao et al. in [ZYZL18] optimized the body block of Valiant's UC by finding a more efficient block using exhaustive search through all possibilities. As opposed to Valiant's UC that uses 15 additional nodes in the body block, their block uses only 14 additional nodes and therefore, their UC achieves an asymptotically better size of



(a) Body block of Valiant [Val76]. (b) Body block of Zhao et al. [ZYZL18].

Figure 4: Body block $B^{(4)}$ alternatives for 4-way EUG $U^{(4)}_n(\Gamma_1)$.

[1] These G' graphs are constructed from $\Gamma_1(n)$ graph G in order to define the embedding, but are not acyclic.
[2] $n \pmod 4$ of these have size $\lfloor \frac{n}{4} - 1 \rfloor$, but for the sake of simplicity, we disregard distinguishing these here.

$\sim 4.5n \log_2 n$. We depict the further optimized body block $B^{(4)}$ of Zhao et al. [ZYZL18] in Fig. 4b. Zhao et al. provide a computer generated proof of that this block can indeed be used to construct universal circuits. Moreover, they show that there exists no block with only 13 additional nodes that can be used to construct UCs in the same manner. This proves that the minimal size of a 4-way UC is the achieved $\sim 4.5n \log_2 n$.

Both body blocks are connected such that the 4 top [bottom] recursion points of one block are the 4 bottom [top] recursion points of the next block. Similarly to the 2-way EUG, 4 sets are created for n nodes, i.e., $R^1_{\lceil \frac{n}{4}-1 \rceil} = \{r^1_1, \ldots, r^1_{\lceil \frac{n}{4}-1 \rceil}\}$, $R^2_{\lceil \frac{n}{4}-1 \rceil} = \{r^2_1, \ldots, r^2_{\lceil \frac{n}{4}-1 \rceil}\}$, $R^3_{\lceil \frac{n}{4}-1 \rceil} = \{r^3_1, \ldots, r^3_{\lceil \frac{n}{4}-1 \rceil}\}$, and $R^4_{\lceil \frac{n}{4}-1 \rceil} = \{r^4_1, \ldots, r^4_{\lceil \frac{n}{4}-1 \rceil}\}$ which are the poles of 4 $U_{\lceil \frac{n}{2} \rceil - 1}(\Gamma_1)$ EUGs in the next recursion step. Then, these also create 4 subgraphs until the recursion base is reached, cf. §3.1.

Theorem 5 ([Val76]). *The resulting 4-way EUG is edge-universal and therefore the resulting circuit is universal.*

The proof of this theorem is analogous to that of Theorem 4.

4 Programming Valiant's Universal Circuits

We designed the detailed embedding algorithm and the open-source UC implementation of [KS16] specifically for the 2-way UC, dealing with the whole UC skeleton as one block. In contrast, based on the modular design of [LMS16], we modularized the edge-embedding task into multiple sub-tasks and described how they can be performed separately in [GKS17]. In this section, we detail this modular approach for edge-embedding a graph into Valiant's ℓ-way EUG, where $\ell = 2$ or $\ell = 4$: the edge-embedding can be split into two parts, which are then combined.

In the following, we describe the two main steps of our modular approach presented in [GKS17] that are based on the edge-embedding algorithm of [KS16]. 1) Block edge-embedding (§4.1) allows for the programming of the blocks visualized in Fig. 3 on p. 12 and in Figs. 4a or 4b on p. 13. 2) Recursion point edge-embedding (§4.2) takes care of the programming of the whole UC. Here, the paths are defined and the necessary information is provided to the blocks (cf. §4.2). The process can be generalized to any 2^i-way EUG. Moreover, the same modular edge-embedding algorithm can be applied with a few modifications for Lipmaa et al.'s generalization to any k-way recursion [LMS16], which we describe later in §5.1.

4.1 Block Edge-Embedding

In this first part of the edge-embedding process, we consider the ℓ top [bottom] recursion points of a block (Figs. 3 and 4a or 4b) as intermediate nodes where the inputs [outputs] of the block enter [leave]. The blocks are built such that any of these inputs can be forwarded to exactly one of the ℓ poles of the block and the output of any pole can be forwarded to exactly one output or another pole having a higher topological order.

We formalize this behaviour as follows: In $U^{(\ell)}_n(\Gamma_1) = (V_U, E_U)$, let $B^{(\ell)}$ be the $(i-1)$-th block in the chain visualized in Fig. 3 for $\ell = 2$ and Fig. 4a or Fig. 4b for $\ell = 4$ with poles $p_{\ell i+1}, \ldots, p_{\ell i+\ell}$. Let the mapping $\eta^U : V_U \to \mathbb{N}^+$ denote a topological order of all nodes and poles in V_U. Then, the nodes $r^1_i, \ldots, r^\ell_i$ and $r^1_{i+1}, \ldots, r^\ell_{i+1}$ denote the input and output recursion points of block $B^{(\ell)}$. Additionally, let $in = (in_1, \ldots, in_\ell) \in \{0, \ldots, \ell\}^\ell$ and $out = (out_1, \ldots, out_\ell) \in \{0, \ldots, 2\ell - 1\}^\ell$ denote the input and output vectors of $B^{(\ell)}$. The value 0 of the input and output vectors is a *dummy value* which is

used if there is no specific path between an input [a pole] and pole [or output] of $B^{(\ell)}$. The output vector has a larger value range, since a pole can be forwarded to another pole or an output recursion point. Therefore, we use values $1,\dots,\ell-1$ for poles $p_{\ell i+2},\dots,p_{\ell i+\ell}$, and values $\ell,\dots,2\ell-1$ for the output recursion points. Pole $p_{\ell i+1}$ cannot be a destination for a path in $B^{(\ell)}$, since $\eta^U(p_{\ell i+1})$ is less than the topological order of any other pole in $B^{(\ell)}$. Additionally, the values of *in* and *out* need to be pairwise different or 0. Every combination of input and output vector covering the conditions formalized below in Eqs. 2–6 are valid for $B^{(\ell)}$. A pair $(r_i^l,p_j)\in\mathcal{P}$ or $(p_j,r_{i+1}^l)\in\mathcal{P}$ is a path from r_i^l to p_j or p_j to r_i^l in the set of all paths $\mathcal{P}$ in $B^{(\ell)}$. Then, $\mathcal{P}_B^{(\ell)}\subseteq\mathcal{P}$ denote the paths that are to be edge-embedded (cf. §3.1).

$$\textsc{InPolePath}:\quad \forall l\in\{1,\dots,\ell\}: in_l\neq 0\rightarrow(r_i^l,p_{\ell i+in_l})\in\mathcal{P}_B^{(\ell)}, \tag{2}$$

$$\textsc{PolePolePath}:\quad out_l\neq 0\wedge out_l<\ell\rightarrow(p_j,p_{\ell i+1+out_l})\in\mathcal{P}_B^{(\ell)}\wedge\eta^U(p_j)<\eta^U(p_{\ell i+1+out_l}), \tag{3}$$

$$\textsc{PoleOutPath}:\quad out_l>\ell-1\rightarrow(p_{\ell i+l},r_{i+1}^{out_l-\ell-1})\in\mathcal{P}_B^{(\ell)}. \tag{4}$$

$$\textsc{InDiff}:\quad \forall in_i,in_j\in in: i\neq j\rightarrow in_i=0\vee in_i\neq in_j. \tag{5}$$

$$\textsc{OutDiff}:\quad \forall out_i,out_j\in out: i\neq j\rightarrow out_i=0\vee out_i\neq out_j. \tag{6}$$

4.2 Recursion Point Edge-Embedding

Block edge-embedding covers only the programming of the nodes within a block. Another task is to program the recursion points. We use the supergraph construction of [KS16] which, in every step, splits a $\Gamma_2(n)$ graph in two $\Gamma_1(n)$ graphs, which are merged to two $\Gamma_2(\lceil\frac{n}{2}-1\rceil)$ graphs. [KS16] use this for defining the paths in Valiant's 2-way EUG. For Valiant's 4-way EUG, we use every second step of their algorithm with a minor modification. We describe our modular algorithm for the 2-way and 4-way UCs in Listing 1.

Let $C_{u,v}^k$ be the Boolean circuit computing function f that our UC needs to compute, and $G\in\Gamma_2(n)$ its graph representation (cf. §2.2).

1. Splitting $G\in\Gamma_2(n)$ in two $\Gamma_1(n)$ graphs G_1 and G_2: As described in §3.1, Valiant's UC is derived from an EUG for $\Gamma_2(n)$ graphs, which consists of two EUGs for $\Gamma_1(n)$ graphs merged by their poles. Therefore, G is split into two $\Gamma_1(n)$ graphs G_1 and G_2. G_1 and G_2 then need to be edge-embedded into EUGs $(U_n^{(\ell)}(\Gamma_1))_1$ and $(U_n^{(\ell)}(\Gamma_1))_2$, respectively. $G=(V,E)\in\Gamma_2(n)$ is split by 2-coloring its edges as described in [Val76, KS16], which can always be done due to Kőnig's theorem [Kő31, LP09b]. After 2-coloring, E is divided to sets E_1 and E_2, using which we build $G_1=(V,E_1)$ and $G_2=(V,E_2)$, with the following conditions:

$$\textsc{EdgeIn}E_1\textsc{or}E_2:\quad \forall e\in E:(e\in E_1\vee e\in E_2)\wedge\neg(e\in E_1\wedge e\in E_2). \tag{7}$$

$$\textsc{Fanin1}E_1:\quad \forall e=(v_1,v_2)\in E_1:\neg\exists e'=(v_3,v_4)\in E_1: v_2=v_4\vee v_1=v_3. \tag{8}$$

$$\textsc{Fanin1}E_2:\quad \forall e=(v_1,v_2)\in E_2:\neg\exists e'=(v_3,v_4)\in E_2: v_2=v_4\vee v_1=v_3. \tag{9}$$

2. Merging a $\Gamma_1(n)$ graph into a $\Gamma_2(\lceil\frac{n}{2}-1\rceil)$ graph: In an EUG, the number of poles decreases in each recursion step and therefore, merging a $\Gamma_1(n)$ graph into a $\Gamma_2(\lceil\frac{n}{2}-1\rceil)$ graph provides information about the paths to be taken. Let $G_1=(V,E)\in\Gamma_1(n)$ be a topologically ordered graph and $G_m=(V',E')\in\Gamma_2(\lceil\frac{n}{2}-1\rceil)$ be a graph with nodes $v'_1,\dots,v'_{\lceil\frac{n}{2}\rceil}$. We define two labellings η_{in} and η_{out} on G_m with $\eta_{\text{in}}(v_i)=i$ and $\eta_{\text{out}}(v_i)=\eta_{\text{in}}(v_i)-1=i-1$. Additionally, we define a mapping θ_V that maps a node $v_i\in V$ to a node $v_j\in V'$ with $\theta_V(v_i)=v'_{\lceil\frac{i}{2}\rceil}$, i.e., two nodes in G_1

are mapped to one node in G_m. At last, we define a mapping θ_E that maps an edge $e_i = (v_i, v_j) \in E$ to an edge $e_j \in E'$ with $\theta_E((v_i, v_j)) = (v_{\eta_{\text{in}}(\theta_V(v_i))}, v_{\eta_{\text{out}}(\theta_V(v_j))})$, i.e., every edge in G_1 is mapped to an edge in G_m as follows: $e = (v_i, v_j) \in E$ is mapped to $e' = (v'_k, v'_l) \in E'$, such that $v'_k = \theta_V(v_i)$, but v'_l is not the new node of v_j in G_m but v'_{l+1}. G_m is built as follows: $V' = \{v'_1, \ldots, v'_{\lceil \frac{n}{2} \rceil}\}$ and $E' = \bigcup_{e \in E} \theta_E(e)$. Then for all $e = (v'_i, v'_j) \in E'$ and $j < i$, e is removed from E', along with the last node $v_{\lceil \frac{n}{2} \rceil}$ (due to the definition of θ_E, it does not have any incoming edges). The resulting G_m is a topologically ordered graph in $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$.

3. The supergraph for Valiant's EUG construction: In the first step, G is split to two $\Gamma_1(n)$ graphs G_1 and G_2. G_1 and G_2 contain all the edges that should be embedded as paths between poles in the first and second EUGs for $\Gamma_1(n)$, respectively. We now explain how to edge-embed the $\Gamma_1(n)$ graph G_1 into an EUG $U_n^{(\ell)}(\Gamma_1)$ (for G_2 it is similar).

For embedding in a 2-way UC, G_1 is firstly merged to a $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$ graph G_m. G_m is then 2-colored and split into two $\Gamma_1(\lceil \frac{n}{2} - 1 \rceil)$ graphs G_1^1 and G_1^2 [KS16]. These get merged to two $\Gamma_2(\lceil \frac{\lceil \frac{n}{2} - 1 \rceil}{2} - 1 \rceil)$ graphs G_m^1 and G_m^2. G_1^1 is the first and G_1^2 is the second subgraph of G_1. Then $G_1^{\psi \circ 1}$ and $G_1^{\psi \circ 2}$ denote the first and second subgraph of G_1^ψ, respectively. These steps are repeated until the recursion base is reached.

In Valiant's *4-way EUG construction* [Val76], a supergraph that creates 4 subgraphs in each step is necessary. We require a merging method where a $\Gamma_1(n)$ graph is merged to a $\Gamma_4(\lceil \frac{n}{4} - 1 \rceil)$ graph where 4 nodes build a new node, and 4-color this graph to retrieve 4 subgraphs. However, this can directly be solved by using the method described above from [KS16]: after repeating the 2-coloring and the merging twice, we gain 4 subgraphs (G_1^{11}, G_1^{12}, G_1^{21} and G_1^{22}). These can be used as if they were the result of 4-coloring the graph obtained by merging every 4 nodes into one.

However, there is a modification in this case: the first 2-coloring is a preprocessing step, which does not map to an EUG recursion step. Therefore, we have to define another labelling $\eta_{\text{out}_P}(v) = \eta_{\text{in}}(v)$, since in this preprocessing step we need to keep node $v_{\lceil \frac{n}{2} \rceil}$. Then the creation of the supergraph for the 4-way EUG construction works as follows: We merge G_1 to a $\Gamma_2(\lceil \frac{n}{2} \rceil)$ graph with labelling η_{in} and η_{out_P} and get G_m. After that, we split G_m into two $\Gamma_1(\lceil \frac{n}{2} \rceil)$ graphs G_1^1 and G_1^2. These get merged to $\Gamma_2(\lceil \frac{n}{4} \rceil - 1)$ graphs G_m^1 and G_m^2 using the η_{in} and η_{out} labellings. Finally, these two graphs get splitted into 4 $\Gamma_1(\lceil \frac{n}{4} - 1 \rceil)$ graphs G_1^{11}, G_1^{12}, G_1^{21} and G_1^{22}. These are the relevant graphs for the first recursion step in Valiant's 4-way EUG construction. Now we continue for all 4 subgraphs until we reach the recursion base.

ℓ-way Edge-Embedding Algorithm. In Listing 1, we combine block edge-embedding and recursion point edge-embedding:

Let $\mathcal{U}$ denote the part of $U_n^{(\ell)}(\Gamma_1)$ without recursion steps (the main skeleton) and $G_1 = (V, E)$ be the $\Gamma_1(n)$ graph which is to be edge-embedded in $U_n^{(\ell)}(\Gamma_1)$. $\mathcal{S}$ denotes the set of ℓ subgraphs of G_1 in the supergraph, i.e. $\mathcal{S} = \{G_1^1, G_1^2\}$ for $\ell = 2$, and $\mathcal{S} = \{G_1^{11}, G_1^{12}, G_1^{21}, G_1^{22}\}$ for $\ell = 4$. A *recursion step graph* of $\mathcal{U}$ is one of the graphs having one of the ℓ sets of recursion points as poles (e.g. $r_1^1, \ldots, r_{\lceil \frac{n}{\ell} - 1 \rceil}^1$) without the recursion steps. $\mathcal{R}$ denotes the set of all ℓ recursion step graphs of $\mathcal{U}$, and $\mathcal{B}$ denotes the set of all blocks in $\mathcal{U}$.

We give a brief explanation of Listing 1 that describes the edge-embedding process. For any edge $e = (v_i, v_j) \in E$ in G_1, b_i and b_j denote the block numbers in which v_i and v_j are. We distinguish between two cases:

Listing 1: Edge-embedding algorithm for Valiant's ℓ-way EUG

```
1  procedure edge−embedding (U, G1 = (V, E))
2   Let S be the set of the ℓ Γ1 subgraphs of G1 in the supergraph
3   Let R be the ℓ recursion step graphs
4   Let B be the set of blocks in U
5   for all e = (vi, vj) ∈ E do
6     Let i' and j' denote the positions of vi and vj in their blocks
7     bi ← ⌈i/ℓ⌉, bj ← ⌈j/ℓ⌉ // number of block in which vi and vj are
8     Let out [r1] denote the output vector [recursion points] of B[bi]
9     Let in [r0] denote the the input vector [recursion points] of B[bj]
10    if bi = bj do // vi and vj are in the same block
11          if vi ≠ vj do
12              out_i' ← j' − 1
13          end if
14    else // vi and vj are in different blocks
15          Let s = (V', E') ∈ S denote the Γ1 graph with e' = (p_bi, p_bj−1) ∈ E' and e' is not marked
16          Mark e'
17          Let x denote the number with s = S[x]
18          Set the control bit of r_0^x to 1
19          if bj = bi + 1 do // bj and bi are neighbours
20              y ← 0
21          else
22              y ← 1
23          end if
24          Set the control bit of r_1^x to y
25          out_i' ← x + ℓ, in_x ← j'
26    end if
27  end for
28  Edge−embed all blocks in B // edge−embed all sub−blocks
29  for i = 1 to ℓ do
30    if S[i] exists do
31     call edge−embedding(R[i], S[i])
32    end if
33  end for
34 end procedure
```

Case 1. v_i and v_j are in the same block: $b_i = b_j$. The edge-embedding is solved within the block and no recursion points have to be programmed for the path. Therefore, vector *out* of block $\mathcal{B}[b_i]$ is set accordingly.

Case 2. v_i and v_j are in different blocks: $b_i \neq b_j$. There exists an edge $e' = (b_i, b_{j-1})$ in one of the ℓ $\Gamma_1(\lceil \frac{n}{\ell} - 1 \rceil)$ subgraphs of G_1 that is not yet used for an edge-embedding. This determines that the path in the next recursion step has to be between poles p_{b_i} and $p_{b_{j-1}}$. We denote with $s \in \mathcal{S}$ the subgraph of G_1 which contains e', and x denotes its number in S, i.e. $\mathcal{S}[x] = s$. This implies in which of the ℓ recursion step graphs we need to edge-embed the path from p_{b_i} to $p_{b_{j-1}}$, and so which recursion points we need to program. We first set the control bit of the x-th input [output] recursion points to 1 since the path between the poles with labelling i and j enters [leaves] the next recursion step over this recursion point. A special case to be considered here is when blocks $\mathcal{B}[b_i]$ and $\mathcal{B}[b_j]$ are neighbours (i.e. $b_j = b_i + 1$). Then, the path enters and leaves the next recursion step graph at the same node, whose control bit thus has to be 0. The output vector of block $\mathcal{B}[b_i]$ is the i'^{th} value to the x^{th} recursion point and the input vector of block $\mathcal{B}[b_j]$ is the x^{th} value to the j'^{th} pole in this block.

We repeat these steps for all edges $e \in E$. Since all input and output vector of all blocks in $\mathcal{B}$ are set, they can be embedded with the block edge-embedding. For all ℓ subgraphs of G_1 in the supergraph and in the EUG, we call the same procedure with $\mathcal{S}[i] \in \mathcal{S}$, $\mathcal{R}[i] \in \mathcal{R}$, $1 \leq i \leq \ell$.

5 Extensions to Valiant's UC Constructions

Here, we describe ideas for novel UC constructions and implementations. Firstly, in §5.1, we describe the k-way generalization of Valiant's UC presented by Lipmaa et al. in [LMS16]. In §5.2, we describe our modular building blocks for a potentially more efficient *3-way UC*. We show that Valiant's optimized $U_3(\Gamma_1)$ cannot directly be applied as a building block in the construction due to the fact that it must have an additional node to be part of a generic EUG. We prove that the EUG without this node is not a valid EUG by showing a counterexample. Therefore, it actually results in a worse asymptotic size than Valiant's 2-way and 4-way UCs [Val76]. Thereafter, in §5.3, we propose a *hybrid UC*, utilizing both Valiant's 2-way and 4-way UCs or Valiant's 2-way and Zhao et al.'s 4-way UC [ZYZL18] so that the overall size of the resulting hybrid UC is minimized, and is at least as efficient as the better construction for the given size (in §6.2 we show its concrete improvement). Finally, in §5.4, we propose a different modular and scalable approach of Valiant's 4-way UC. This approach requires a lot of modifications in the UC generation and programming algorithm, but can be generalized to any k-way UC or to our hybrid UC.

5.1 Generalized k-Way UC

In [LMS16], Lipmaa et al. generalize Valiant's approach by providing a UC with any number of recursion points k, the so-called *k-way EUG or UC*. We note that their construction slightly differs from Valiant's EUG, since they do not consider the restriction on the fanout of the poles, i.e., the nodes in the EUG that correspond to universal gates or inputs (cf. §3.1). This optimization has also been included in [KS16] when translating an EUG to a UC, but including it in the block design leads to better sizes for the number of XOR gates. This, however, does not make a difference in case of our most prominent application of private function evaluation (PFE) (cf. §1.1), where XOR gates are free, i.e., do not require cryptographic operations and communication.

The idea is to split $n = u + v + g$ in $m = \lceil \frac{n}{k} \rceil$ blocks as shown in Fig. 5. Every block i consists of k inputs $r_i^1, r_i^2, \ldots, r_i^k$ and k outputs $r_{i+1}^1, r_{i+1}^2, \ldots, r_{i+1}^k$ as well as k poles, except for the last block which has a number of poles depending on n mod k. For every $j \leq k$, the list of all r_i^j builds the poles of the j^{th} subgraph of the next recursion step, i.e., we have k subgraphs. Additionally, every block begins and ends with a Waksman permutation network [Wak68] such that the inputs and outputs can be permuted to any pole. A Y-switching block is placed in front of every pole p_i which is connected to the i^{th} output of the permutation network as well as the i^{th} output of a block-intern EUG $U_k(\Gamma_1)$. This means that [LMS16] reduce the problem of finding an efficient k-way EUG $U_n^{(k)}(\Gamma_2)$ block $B^{(k)}$ to the problem of finding the smallest EUG $U_k(\Gamma_1)$. Their solution is to build the block-intern EUG with the UC of [KS08b], which was claimed to be more efficient for smaller circuits than [Val76]. Moreover, they calculate the optimal k value to be around 3.147 with their construction, which implies that the best solutions are found using small EUGs, for which Valiant provides hand-optimized solutions (i.e., for $k = 2, 3, 4, 5, 6$) [Val76].

We note that the results recently presented by Zhao et al. [ZYZL18] do not fit into this generalized k-way construction. Therefore, Zhao et al.'s optimized 4-way block is an optimization over Valiant's modular 4-way block construction [Val76].

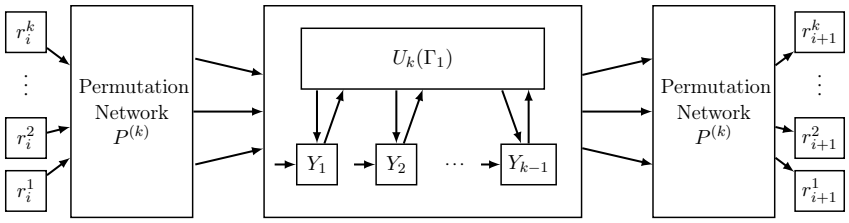


Figure 5: k-way EUG construction $U_n^{(k)}(\Gamma_1)$ [LMS16].

Programming the Generalized UC

In this section, we extend the recent work of [LMS16] by providing a detailed and modular embedding mechanism for any k-way EUG construction. We provide the main differences to the edge-embedding of the 2-way and 4-way EUG detailed in §4.

k-way Block Edge-Embedding. In this setting, our main block is a programmable block $B^{(k)}$ with k poles $p_1, \ldots, p_k$, and k input [output] recursion points $r_0^1, \ldots, r_0^k$ $[r_1^1, \ldots, r_1^k]$. $B^{(k)}$ is topologically ordered with mapping η^U as defined in §2.1. Vectors $in = (in_1, \ldots, in_k) \in \{0, \ldots, k\}^k$, and $out = (out_1, \ldots, out_k) \in \{0, \ldots, 2k-1\}^k$ denote the input and output vectors of $B^{(k)}$, respectively. Values $k, \ldots, 2k-1$ in out denote the recursion point targets $r_1^1, \ldots, r_1^k$ (cf. §4.1). The setting of in and out is formalized in Eqs. 2–6 when $\ell = k$.

k-way Recursion Point Edge-Embedding. $G \in \Gamma_2(n)$ denotes the transformed graph of a Boolean circuit $C_{u,v}^g$, where $n = u + v + g$.

1. Splitting $G \in \Gamma_2(n)$ in two $\Gamma_1(n)$ graphs G_1 and G_2: Similarly as in §4.2, we first split G into two $\Gamma_1(n)$ graphs G_1 and G_2 with 2-coloring.

2. Merging a $\Gamma_1(n)$ graph into a $\Gamma_k(\lceil \frac{n}{k} - 1 \rceil)$ graph: $G_1 = (V, E) \in \Gamma_1(n)$ is merged into a $\Gamma_k(\lceil \frac{n}{k} - 1 \rceil)$ graph $G_m = (V', E')$ (same for G_2). Therefore, we redefine mapping θ_V (cf. §4.2) that maps node $v_i \in V$ to node $v_j \in V'$. In this scenario, k nodes in V build one node in V', so $\theta_V(v_i) = v_{\lceil \frac{i}{k} \rceil}$. The mapping of the edges θ_E is the same as in the 2-way and 4-way EUG construction, and $(v_i', v_j') \in E'$ where $j < i$ edges are removed along with $v_{\lceil \frac{n}{k} \rceil}$ in the end. G_m is then a topologically ordered graph in $\Gamma_1(\lceil \frac{n}{k} - 1 \rceil)$.

3. The supergraph for Lipmaa et al.'s k-way EUG construction: The next step of the construction is to split $G_m \in \Gamma_1(\lceil \frac{n}{k} - 1 \rceil)$ into k $\Gamma_1(\lceil \frac{n}{k} - 1 \rceil)$ graphs. This is done with k-coloring: a directed graph $K = (V, E)$ can be k-colored, if k sets $E_1, \ldots, E_k \subseteq E$ cover the following conditions:

$$\textsc{Disjoint} \qquad \forall i, j \in \{1, \ldots, k\} : i \neq j \rightarrow E_i \cap E_j = \emptyset. \tag{10}$$

$$\textsc{EdgeIn}E_i \qquad \forall e \in E : \exists i \in \{1, \ldots, k\} : e \in E_i. \tag{11}$$

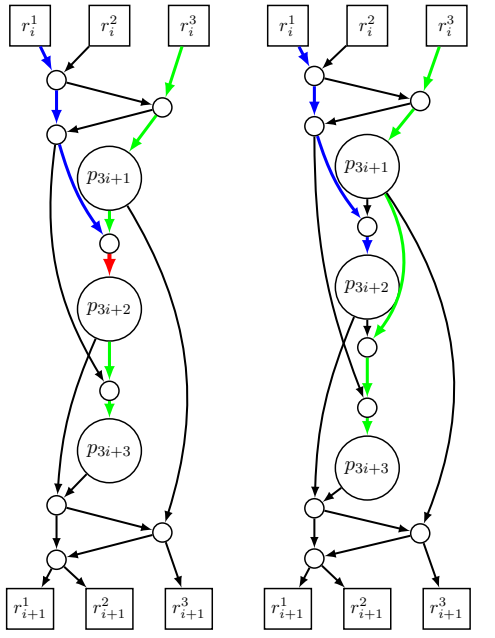
$$\textsc{Fanin1}E_i \qquad \forall i \in \{1, \ldots, k\}, \forall e = (v_1, v_2) \in E_i : \neg \exists e' = (v_3, v_4) \in E_i \setminus \{e\} : v_2 = v_4 \vee v_1 = v_3. \tag{12}$$

According to Kőnig's theorem [Kő31, LP09b] described in §2.1, $\Gamma_k(n)$ graphs can always be k-colored efficiently with a dedicated algorithm. The rest of the supergraph construction and the way it is used for edge-embedding is the same as for the 2-way and 4-way EUG as described in §4.2.

k-way Edge Embedding Algorithm. The edge-embedding algorithm is the same as shown in Listing 1, with $\ell = k$.

5.2 Potentially More Efficient 3-Way UC

The optimal k value for minimizing the size of the k-way UC was calculated to be 3.147 in [LMS16]. We describe our idea of a 3-way UC. Intuitively, based on an optimization by Valiant [Val76], this UC should result in the best asymptotic size. The asymptotic size of any k-way UC depends on the size of its modular body block $B^{(k)}$ (e.g., Figs. 4a or 4b on p. 13 for the 4-way UC). Once it is determined, the size of the UC is $\text{size}(U_n^{(k)}(\Gamma_2)) = 2 \cdot \text{size}(U_n^{(k)}(\Gamma_1)) \sim 2 \cdot \frac{\text{size}(B^{(k)})}{k} n \log_k n = 2 \cdot \frac{\text{size}(B^{(k)})}{k \log_2(k)} n \log_2 n$. The modular block consists of two permutation networks $P^{(k)}$, an EUG $U_k(\Gamma_1)$, and $(k-1)$ Y-switching blocks (cf. §5.1, [LMS16])[3].



(a) Body Block with Valiant's $U_3(\Gamma_1)$ (b) Body Block with our generic $U_3(\Gamma_1)$

Figure 6: Body block $B^{(3)}$ construction for our 3-way EUG $U_n^{(3)}(\Gamma_1)$.

Size of Body Block $B^{(3)}$ with Valiant's Optimized $U_3(\Gamma_1)$. According to Valiant [Val76], an EUG $U_3(\Gamma_1)$ with 3 poles contains only 3 connected poles (used as recursion base in §3.1). An optimal permutation network $P^{(3)}$ that achieves the lower bound has 3 nodes as well. This implies that $\text{size}(B^{(k)}) = 2 \cdot P^{(3)} + \text{size}(U_3(\Gamma_1)) + (3-1) = 11$. Then, the size of the UC becomes $\sim 2 \cdot \frac{11}{3 \log_2 3} n \log_2 n \sim 4.627 n \log_2 n$, which means an asymptotically by around 2.5% smaller size than that of Valiant's 4-way UC with $\sim 4.75 n \log_2 n$.

However, there is a flaw in this initial design. Valiant's $U_3(\Gamma_1)$ only works as an EUG for 3 nodes under special conditions, e.g., when it is a subgraph within a larger EUG. There are 3 possible edges in a topologically ordered graph $G = (V, E)$ in $\Gamma_1(3)$: $(1,2), (2,3)$ and $(1,3)$. $(1,2)$ and $(2,3)$ can be directly embedded in $U_3(\Gamma_1)$ using (p_1, p_2) and (p_2, p_3), respectively. $(1,3)$, however, has to be embedded as a path *through* node 2, i.e., as a path $((p_1, p_2), (p_2, p_3))$. When $U_3(\Gamma_1)$ is a subgraph of a bigger EUG, this is possible by programming p_2 accordingly. However, when we use this $U_3(\Gamma_1)$ as a building block in our EUG, it cannot directly be applied, due to the fact that the programming of p_2 depends on other constraints as well. A generic $U_3(\Gamma_1)$ that can embed $(1,3)$ without going through p_2 as before has an additional Y-switching block between p_2 and p_3.

We depict in Fig. 6a the 3-way body block that uses Valiant's optimized $U_3(\Gamma_1)$ in the k-way block design of [LMS16]. Assume that the output of pole p_{3i+1} has to be directed to pole p_{3i+3} (green path). Then, it needs to go through pole p_{3i+2}, which means that the red edge going to p_{3i+2} is used by this path. However, there can be an other edge coming from the permutation network as an input to p_{3i+2}, e.g., from p_{3i} from the preceding block through r_i^1 (blue path). This cannot be directed to p_{3i+2} anymore, as shown in Fig. 6a, since the red edge would carry two different values. Therefore, in the 3-way body block construction, it does not suffice to use Valiant's optimized $U_3(\Gamma_1)$ [Val76].

[3] We note that in this section, we design the body block according to [LMS16], i.e., the poles do not have a fanout restriction. However, all other nodes have fanout-2 restriction.

Listing 2: Hybrid construction algorithm, where $B^{(k)}(i)$, $H^{(k)}(i)$ and $T^{(k)}(i)$ denote body, head and tail blocks with i poles in the k-way UC, respectively.

procedure hybrid $(p_1, \ldots, p_n,\ K = \{2, 4\})$
  Let size$(U_{n'}^{\text{hybrid}(K)}(\Gamma_1))$ be the function calculating the size of the smaller hybrid constructions with size $n' \leq n$
  for all $k \in K$ **do** *// Number of poles in the last block for all k*
    if $n \mid k$ **do**
      $m_k \leftarrow k$
    else
      $m_k \leftarrow n \mod k$
    end if
    $s_k \leftarrow \text{size}(H^{(k)}(k)) + \left(\lceil \frac{n}{k} \rceil - 3\right) \cdot \text{size}(B^{(k)}(k)) + \text{size}(B^{(k)}(r_k)) + \text{size}(T^{(k)}(m_k)) + m_2 \cdot \text{size}\left(\text{size}(U^{\text{hybrid}(K)}_{\lceil \frac{n}{2} - 1 \rceil}(\Gamma_1))\right) + ((k - m_k) \cdot \text{size}\left(\text{size}(U^{\text{hybrid}(K)}_{\lfloor \frac{n}{k} - 1 \rfloor}(\Gamma_1))\right)$
  end for
  $s_i \leftarrow \min(s_k : k \in K)$ *// Choose the better construction*
  // GENERATION
  Create skeleton **for** i–way construction with n poles
  call hybrid$\left(r_1^1, \ldots, r^1_{\lceil \frac{n}{i} - 1 \rceil}, K\right)$, ..., hybrid$\left(r_1^{m_i}, \ldots, r^{m_i}_{\lceil \frac{n}{i} - 1 \rceil}, K\right)$
  if $(i - m_i) > 0$ **do call** hybrid$\left(r_1^{m_i}, \ldots, r^{m_i}_{\lfloor \frac{n}{i} - 1 \rfloor}, K\right)$, ..., hybrid$\left(r_1^{i}, \ldots, r^{i}_{\lfloor \frac{n}{i} - 1 \rfloor}, K\right)$
  end if
  // PROGRAMMING
  Call edge–embedding$(\mathcal{U}^{(i)}, G_1^{(i)} = (V, E))$ *//Call embedding algorithm corresponding to i*
end procedure

Size of Body Block $B^{(3)}$ with Our Generic $U_3(\Gamma_1)$. In Fig. 6b, we show the 3-way body block with the generic $U_3(\Gamma_1)$ that allows the output from p_{3i+1} to be directed to p_{3i+3} without having to go *through* p_{3i+2} (green path), and the edge going into p_{3i+2} can be utilized by the path directed into this node (blue path). This results in $\text{size}(B^{(3)}) = 2 \cdot P^{(3)} + \text{size}(U_3(\Gamma_1)) + (3 - 1) = 12$, which implies that the size of the UC is $\sim 2 \cdot \frac{12}{3 \log_2 3} n \log_2 n = 5.047 n \log_2 n$. Unfortunately, this is worse than the size of the 2-way UC with $\sim 5 n \log_2 n$, and we therefore conclude that the most efficient known UC is Valiant's 4-way UC with Zhao et al.'s optimization.

Recently, Zhao et al. [ZYZL18] have shown by exhaustive search over all possible topologies that the 3-way body block $B^{(3)}$ presented in Fig. 6b results in the smallest 3-way UC by showing that no block with only 11 additional nodes can be used as a universal block, and indeed, our block with 12 additional nodes can be utilized.

5.3 2/4 Hybrid UC Construction

In this section, we detail our hybrid UC that minimizes its size based on Valiant's 2-way and 4-way UCs with the optimization by Zhao et al. [ZYZL18], which yields the smallest UCs to date. Given the size of the input circuit $C^g_{u,v}$, i.e., $n = u + v + g$, we can calculate at each recursion step if it is better to create 2 subgraphs of size $\lceil \frac{n}{2} - 1 \rceil$ and utilize the 2-way recursive skeleton, or it is more beneficial to create a 4-way recursive skeleton with 4 subgraphs of size $\lceil \frac{n}{4} - 1 \rceil$.

We assume that for every n, we have an algorithm that computes the size (i.e., $\text{size}(U_n^{\text{hybrid}(K)}(\Gamma_1))$) of the hybrid UC for sizes smaller than n. We give details on how it is computed in §6. Then, Listing 2 describes the algorithm for constructing a hybrid UC, at each step based on which strategy is more efficient. We note that our hybrid construction is generic, and given multiple k-way UCs as parameter K ($K = \{2, 4\}$ in our example), it minimizes the concrete size of the resulting UC.

5.4 Scalable 4-Way UC Construction

Our existing implementations of [KS16, GKS17] store the whole UC of size $\mathcal{O}(n \log n)$ in memory, which therefore becomes a bottleneck when it comes to scalability. In this section, we present the design of our scalable universal circuit construction. Specifically, we show how Valiant's 4-way UC can be modified to use $\mathcal{O}(n)$ memory in the input circuit size n at each step of the execution. We note that our approach is generic and with additional implementation effort, it can be extended to any k-way UC as well as for the 4-way UC of Zhao et al. [ZYZL18].

In this section, we present our design that utilizes two separate phases. The first phase is *scalable UC generation* (§5.4.1), where the universal circuit is generated given the size n of the input circuit. This is solved by generating the topologically ordered UC layer by layer, each of which has size $\mathcal{O}(n)$. The output of this step is a set of circuit files, which all contain a subgraph of size $\mathcal{O}(n)$, which helps to significantly reduce the complexity of the second phase, i.e., *scalable UC programming* (§5.4.2). In this step, the subcircuits resulting from the first phase are programmed individually, i.e., we proceed subcircuit by subcircuit instead of edge by edge of the input circuit as before. Therefore, the output of this step is a set of programming files that contain the programming bits respective to the circuit files. In §7.2, we will show experimentally that our scalable UC construction significantly reduces the memory usage.

5.4.1 Scalable Per-Block UC Generation

The underlying idea behind our scalable UC generation is to generate the blocks of the main skeleton one by one, only keeping one such block and its corresponding subgraph nodes in memory at once. In this scenario, these blocks will be regarded as layers. Additionally, we store some necessary information from the preceding three layers in dedicated files, but delete these as soon as they become redundant. The required additional information is the topological order of nodes that are already defined and have edges directed into the current layer. Since the number of subgraphs in any layer is $\mathcal{O}(n)$, the number of nodes held in memory at any point is $\mathcal{O}(n)$ as well, since in each layer there are only a constant number of nodes.

Our scalable UC generation relies on the fact that at each block of the main skeleton, based on the modulo 4 result for each next recursion step, we know which part of the next subgraph skeleton or potentially recursion base graph we build at each layer. This observation helps us reconstruct how the subgraphs may look like for a given body block in Valiant's 4-way UC. Since the structure of this is complicated and there are many cases to consider, we show in Fig. 7 the cases for Valiant's body block from Fig. 4a on p. 13 [Val76], and note that head and tail blocks can be constructed analogously. Moreover, a similar scalable design can be constructed for Zhao et al.'s body block (Fig. 4b) [ZYZL18].

Fig. 7d shows a recursive block construction with Figs. 7a – 7c being base cases. From Fig. 7, each body block construction type is denoted by B^i where $i = \{0, 1, 2, 3\}$[4] is the position of nodes between two poles in a body block in the subgraph. A given subgraph has node(s) between every two set of recursion points of the parent graph to which this subgraph belongs. We know that the recursion points, for instance $\{r_1^1, ..., r_{\lceil \frac{n-4}{4} \rceil}^1\}$ are the poles of the next recursion step subgraph. Analogously, we can design head H^i, tail T_x^i, and special last body blocks B_x^i, where $x = \{1, 2, 3, 4\}$ denotes the type of the body or tail block based on the number of input or output recursion points,

[4]Note that our design corresponds to Valiant's 4-way UC, but for simplicity, we use B^i instead of $B^{(4)i}$.

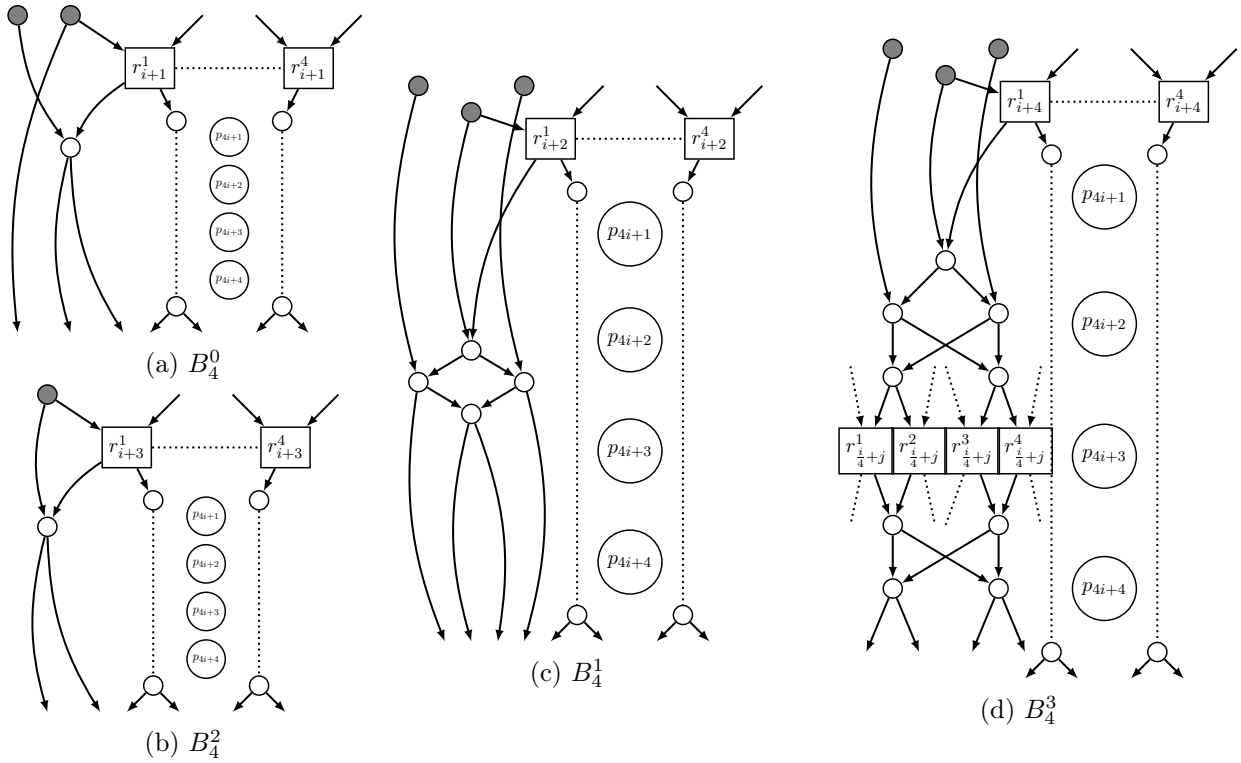


Figure 7: Scalable body block construction. 7a shows the first part $B^0 = B^0_4$ of the body block, 7c the second $B^1 = B^1_4$, 7b the third $B^2 = B^2_4$, and 7d the forth $B^3 = B^3_4$, where further subgraphs are created. We note that the nodes are shown only for one of the four subgraphs, but they are the same for all four subgraphs. Scalable head and tail blocks are designed analogously.

respectively. In the following, we use an example to detail how our scalable UC generation works. We depict the resulting UC files and what their content is in Table 1.

Generation of first (main) skeleton. Generating the first (main) skeleton of the two $U_n(\Gamma_1)$ EUGs that are merged into a $U_n(\Gamma_2)$ EUG differs from the next, recursive steps. Let us consider an example of a DAG with $n = u + k + v = 36$. Ideally, our approach constructs twice the same block from the left and right $U_n(\Gamma_1)$ EUGs. In this scenario for $U_n(\Gamma_1)$, we have one (merged) head block H, seven (merged) body blocks B, and one (merged) tail block T_4 with 4 nodes in the main skeleton. Constructing the first head block is straightforward according to [GKS17, Fig. 4e] as we do not have to construct any subgraph. Thereafter, we construct seven body blocks according to Fig. 4a, and a tail block according to [GKS17, Fig. 4f]. However, these merged blocks require constructing the subgraph nodes in the same layer alongside with it, as we describe next. Note that in this first step, we actually generate twice the four sets of subgraph nodes, since the two $U_n(\Gamma_1)$ EUGs are merged into a $U_n(\Gamma_2)$ EUG (cf. §3.1), but in later recursion steps, only four sets of subgraph nodes are generated.

Generating subgraph nodes recursively per layer. We can generate the subgraph nodes recursively for all recursion steps at a given position for nodes n. In our example with $n = 36$, we only have a head and a tail block for the recursion graph with $\lceil \frac{n-4}{4} \rceil = 8$ poles. Therefore, we construct the first body block with H^0 as subgraph level, the second body block with H^1,

f^{44}	f^{43}	f^{42}	f^{41}	f^4	$\dots$	f^{14}	f^{13}	f^{12}	f^{11}	f^1	f^0	g^1	g^{11}	g^{12}	g^{13}	g^{14}	$\dots$	g^4	g^{41}	g^{42}	g^{43}	g^{44}
					$\dots$						H						$\dots$					
				H^0	$\dots$					H^0	B	H^0					$\dots$	H^0				
				H^1	$\dots$					H^1	B	H^1					$\dots$	H^1				
				H^2	$\dots$					H^2	B	H^2					$\dots$	H^2				
R_1^0	R_1^0	R_1^0	R_1^0	H^3	$\dots$	R_1^0	R_1^0	R_1^0	R_1^0	H^3	B	H^3	R_1^0	R_1^0	R_1^0	R_1^0	$\dots$	H^3	R_1^0	R_1^0	R_1^0	R_1^0
				T_4^0	$\dots$					T_4^0	B	T_4^0					$\dots$	T_4^0				
				T_4^1	$\dots$					T_4^1	B	T_4^1					$\dots$	T_4^1				
				T_4^2	$\dots$					T_4^2	B	T_4^2					$\dots$	T_4^2				
				T_4^3	$\dots$					T_4^3	T	T_4^3					$\dots$	T_4^3				
1	1	1	1	8	$\dots$	1	1	1	1	8	36	8	1	1	1	1	$\dots$	8	1	1	1	1

Table 1: Files storing the UC in our scalable UC generation for an example with $n = 36$.

thereafter H^2 and H^3. The fifth body block is constructed with T^0, the sixth and seventh with T^1 and T^2, respectively, and the tail block with T^3. Recursive scalable blocks are H^3 and B^3 as shown in Fig. 7d. T_4^3 does not have recursion points anymore, since a tail block has no output recursion points. For $n = 8$, we reach a recursion base with $\lceil \frac{n-4}{4} \rceil = 1$. However, for a larger n, more recursion steps might be necessary. Therefore, at each layer, we generate all subgraph nodes necessary and if a recursion step, i.e., H^3 or B^3 occurs, we generate the nodes of the next subgraph as well, etc. We denote the recursion bases by R_1, R_2, R_3 and R_4 with 1, 2, 3, and 4 nodes, respectively.

With this, we have shown how to generate topologically ordered universal circuits using the file system and achieve a scalable algorithm for UC generation that stores at most $\mathcal{O}(n)$ information in memory. Moreover, our approach requires $4.75n \log_2 n$ disk space to store the universal circuit as before, and additionally $\mathcal{O}(n)$ extra storage space for every layer. However, we only store additional data for the prior three layers, and delete any other stored data at each step. In the end of the UC generation, we can delete any additionally stored data. The maximum storage requirement for our algorithm is before deleting the additionally stored data for the last layer, since the size of the UC dominates the storage requirements at any other step (when only a part of it is generated yet).

5.4.2 Scalable UC Programming

As described in §5.4.1, we design our scalable UC generation such that each subgraph is written into a separate file. This is important to also allow the programming step to require only $\mathcal{O}(n)$ memory. It can be observed in Listing 1 on p. 17 that the recursion point edge-embedding algorithm inherently handles the UC subgraph by subgraph (cf. §4.2), which in turn calls the block edge-embedding for all blocks in a subgraph. We observe that each skeleton can be programmed based on the information stored only in the corresponding Γ_1 graph, and therefore, we can store the programming bits in a separate file for each subgraph in the same order as the nodes of the subgraph.

After reading a subgraph from its file resulting from the UC generation step detailed in §5.4.1, it is programmed as described in Listing 1. The embedding starts from the main skeleton in file f^0, and continues with $f^1, \dots, f^4$ and $g^1, \dots, g^4$, etc., and results in the corresponding programming files p^0, $p^1, \dots, p^4$ and $q^1, \dots, q^4$, etc.

6 Size and Depth of UCs

In this section, we review the size and depth of the UCs considered in this article. The *size of the edge-universal graph* $U_n^{(k)}(\Gamma_1)$ is the number of nodes, counting all the poles and nodes created using Valiant's construction §3.1. The *depth of the edge-universal graph* is the number of nodes on the longest path between any two nodes, i.e., essentially the path between the first input and last output. $U_n^{(k)}(\Gamma_2)$ is built from two $U_n^{(k)}(\Gamma_1)$ edge-universal graphs as described in §3.1. When transforming $U_n^{(k)}(\Gamma_2)$ into a UC, the first u poles are associated with inputs, the last v poles with outputs, and the g poles between are realized with universal gates (cf. Eq. 1) whose programming is defined by the corresponding gates in the simulated circuit. The rest of the nodes of $U_n^{(k)}(\Gamma_2)$ are translated into universal (X and Y) switches, whose programming is defined by the edge-embedding of the graph of the circuit G into $U_n^{(k)}(\Gamma_2)$. Thus, when considering the *sizes and depths of the UCs*, we realize the nodes and poles as circuit building blocks and express the concrete and asymptotic sizes in the number of switches and universal gates (cf. §3.2).

In §6.1, we recapitate the asymptotic size and depth of Valiant's 2-way and 4-way UCs [Val76], i.e., $\mathrm{UC}^{\text{Valiant-2}}$ and $\mathrm{UC}^{\text{Valiant-4}}$, respectively, of Zhao et al.'s 4-way UC $\mathrm{UC}^{\text{Zhao et al.-4}}$ [ZYZL18] and of the smallest k-way UCs following Lipmaa et al.'s generalization [LMS16]. Thereafter, in §6.2, we present optimizations that reduce the size (and potentially the depth as well) of UCs, regardless of which constructions were used for their generation. We revise the concrete sizes and depths of $\mathrm{UC}^{\text{Valiant-2}}$ and $\mathrm{UC}^{\text{Valiant-4}}$, $\mathrm{UC}^{\text{Zhao et al.-4}}$ as well as that of our 2/4 hybrid UCs $\mathrm{UC}^{\text{H(Valiant-2,4)}}$ and $\mathrm{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ (cf. §5.3).

6.1 Asymptotic Size and Depth of k-Way UCs

Lipmaa et al.'s k-way UC [LMS16] is discussed briefly in §5.1 and is depicted in Fig. 5 on p. 19. They show that a k-way body block may consist of two permutation networks $P^{(k)}$, an EUG for k nodes, i.e., $U_k(\Gamma_1)$, and additionally, $(k-1)$ Y-switching blocks. In this section, we recapitulate the sizes in Table 2 and depths in Table 3 of these building blocks and give an estimate for the leading constant for Lipmaa et al.'s k-way EUGs and UCs with size $\mathcal{O}(n \log_2 n)$ and depth $\mathcal{O}(n)$, for $k \in \{2, \ldots, 8\}$. We conclude that among all UCs following this generalization, the best size is achieved by Valiant's 4-way UC $\mathrm{UC}^{\text{Valiant-4}}$. This does not exclude the possibility for a more efficient UC, as has been shown in [ZYZL18], where Zhao et al. propose a 4-way UC $\mathrm{UC}^{\text{Zhao et al.-4}}$ using a smaller body block. Therefore, their construction achieves the smallest asymptotic size to date. However, Zhao et al. state that their method most likely cannot be adapted to find more efficient UCs for $k > 4$, since it includes an exhaustive search which becomes too large for this domain.

6.1.1 Edge-Universal Graph with k Poles

Size. Valiant optimized EUGs up to size 6 by hand in [Val76]: for $k = 2$, $U_2(\Gamma_1)$ has two poles, for $k = 3$ we discussed in §5.2 that an additional node is necessary. For $k \in \{4, 5, 6\}$ the sizes are $\{6, 10, 13\}$, as shown in [KS16, Fig. 1] (the nodes denoted as empty circles disappear in the UC). For $k = 7$ and $k = 8$, we observe that $\mathrm{UC}^{\text{Valiant-2}}$ results in a better size than that of $\mathrm{UC}^{\text{Valiant-4}}$ due to the smaller permutation network and less recursion nodes. Therefore, we use these constructions to compute the size of $U_7(\Gamma_1)$ and $U_8(\Gamma_1)$. As mentioned in [LMS16], another possibility is to use the UC of [KS08b] instead of these EUGs since they have better sizes for small circuits. These UCs

k	Ref.	$U_k(\Gamma_1)$	U_k(KS08)	$P_{\text{I}}^{(k)}$	$P_{\text{W}}^{(k)}$	$B^{(k)}$	$U_n^{(k)}(\Gamma_1)$	UC
		#nodes	#nodes	#nodes	#nodes	#nodes	#nodes ($\cdot n \log_2 n$)	#switches ($\cdot n \log_2 n$)
2	[Val76]	**2**	2	1	**1**	5	=2.500	=5.000
3	[GKS17]	**4**	6	3	**3**	12	≈2.524	≈5.047
4	[Val76]	**6**	7	5	**5**	19	=2.375	=4.750
5	[LMS16]	**10**	11	7	**8**	30	≈2.584	≈5.168
6	[LMS16]	**13**	14	10	**11**	40	≈2.579	≈5.158
7	[LMS16]	**19**	19	13	**14**	53	≈2.697	≈5.394
8	[LMS16]	23	**21**	16	**17**	62	≈2.583	≈5.167
4*	[ZYZL18]		-			18	=2.250	=4.500

Table 2: Leading term of the asymptotic $\mathcal{O}(n \log_2 n)$ **sizes** of k-way edge-universal graphs ($U_n^{(k)}(\Gamma_1)$) and universal circuits (UC) and the concrete size of their building blocks for $k \in \{2, \ldots, 8\}$ according to the design of [LMS16]. 4* [ZYZL18] denotes the 4-way construction with the optimized block of [ZYZL18], i.e., UC$^{\text{Zhao et al.-4}}$. n denotes the size of the input $\Gamma_2(n)$ circuit, $U_k(\Gamma_1)$ Valiant's edge-universal graph with k poles, U_k^{KS08} the UC of [KS08b], $P_{\text{I}}^{(k)}$ the permutation network for k nodes achieving the lower bound for the size, and $P_{\text{W}}^{(k)}$ Waksman's permutation network [Wak68]. $B^{(k)}$ is the k-way body block with the best existing alternative for universal circuits and permutation networks marked in bold.

U_k^{KS08} are built from two smaller $U_{\frac{k}{2}}^{\text{KS08}}$, a $P^{(\frac{k}{2})}$ and $\frac{k}{2}$ Y switches [KS08b]. It results in a smaller size of 21 for $k = 8$.

Depth. The depth of the hand-optimized EUGs for $k \in \{2, 3, 4, 5, 6\}$ are respectively $\{2, 4, 5, 7, 10\}$ as shown in [KS16, Fig. 1]. The depth of $U_7(\Gamma_1)$ and $U_8(\Gamma_1)$ becomes respectively 16 and 19 with Valiant's 2-way UC, and 14 and 16 with the UC from [KS08b].

6.1.2 Permutation Networks $P^{(k)}$

Size. Waksman in [Wak68] showed that the lower bound for the size of a permutation network is $\lceil \log_2(k!) \rceil$ for k elements. We present this lower bound in Table 2 as $P_{\text{I}}^{(k)}$. The smallest existing permutation network is Waksman's permutation network $P_{\text{W}}^{(k)}$ [Wak68, BD02]. For $k \in \{2, 3, 4\}$ its size matches the lower bound, but for larger values of k, $P_{\text{W}}^{(k)}$ utilizes additional nodes.

Depth. The depth of a permutation network has lower bound $\lceil \log_2(k!) \rceil + 1$, since each input has to have a path to each output, where switches have only two inputs and two outputs. We show these as the depth of $P_{\text{I}}^{(k)}$ in Table 3. Waksman's permutation network matches the lower bound when $k \in \{2, 3, 4\}$, but utilizes additional nodes for larger values of k.

k	Ref.	$U_k(\Gamma_1)$	U_k(KS08)	$P_1^{(k)}$	$P_W^{(k)}$	$B^{(k)}$	$U_n^{(k)}(\Gamma_1)$	UC
		#nodes	#nodes	#nodes	#nodes	#nodes	#nodes ($\cdot n$)	#switches ($\cdot n$)
2	[Val76]	**2**	2	1	**1**	6	=3.000	=3.000
3	[GKS17]	**4**	5	3	**3**	13	∼4.333	∼4.333
4	[Val76]	**5**	6	3	**3**	15	=3.750	=3.750
5	[LMS16]	**7**	9	4	**5**	22	=4.400	=4.400
6	[LMS16]	**10**	12	4	**5**	26	∼4.333	∼4.333
7	[LMS16]	16	**14**	4	**5**	31	∼4.429	∼4.429
8	[LMS16]	19	**16**	4	**5**	34	=4.250	=4.250
4*	[ZYZL18]		-			14	=3.500	=3.500

Table 3: Leading terms of the asymptotic $\mathcal{O}(n)$ **depths** of k-way edge-universal graphs ($U_n^{(k)}(\Gamma_1)$) and universal circuits (UC) and the concrete depth of their building blocks for $k \in \{2, \ldots, 8\}$ according to the design of [LMS16]. 4* [ZYZL18] denotes the 4-way construction with the optimized block of [ZYZL18], i.e., UC$^{\text{Zhao et al.-4}}$. n denotes the size of the input $\Gamma_2(n)$ circuit, $U_k(\Gamma_1)$ Valiant's edge-universal graph with k poles, U_k^{KS08} the UC of [KS08b], $P_1^{(k)}$ the permutation network for k nodes achieving the lower bound for the depth, and $P_W^{(k)}$ Waksman's permutation network [Wak68]. $B^{(k)}$ is the k-way body block with the best existing alternative for universal circuits and permutation networks marked in bold.

6.1.3 Body Blocks

A body block $B^{(k)}$ is built of $(k-1)$ Y-switching blocks, an EUG for k nodes, and two permutation networks $P^{(k)}$ [LMS16] (cf. Fig. 5). $B^{(k)}$ shown in Tables 2 and 3 is built using Waksman's permutation network $P_W^{(k)}$.

Size. The size of the body block is the sum of the sizes of its building blocks, i.e., $\text{size}(B^{(k)}) = \min\left(\text{size}(U_k(\Gamma_1)), \text{size}(U_k^{\text{KS08}})\right) + 2 \cdot \text{size}(P^{(k)}) + (k-1) \cdot \text{size}(Y)$.

Depth. The depth of $B^{(k)}$ is the number of edges in its building blocks, the additional edges between the different blocks and the recursion nodes. This means that in total $\text{depth}(B^{(k)}) = \min\left(\text{depth}(U_k(\Gamma_1)), \text{depth}(U_k^{\text{KS08}})\right) + 2 \cdot \text{depth}(P^{(k)}) + (k-1) \cdot \text{depth}(Y) + 1$.

6.1.4 Edge-Universal Graphs and Universal Circuits with n Poles

Two k-way EUGs $U_n^{(k)}(\Gamma_1)$ graphs build up an EUG $U_n^{(k)}(\Gamma_2)$ as described in §3.1.

Size. The asymptotic size of EUG $U_n^{(k)}(\Gamma_1)$ is determined as $\text{size}(U_n^{(k)}(\Gamma_1)) = \frac{\text{size}(B^{(k)})}{k \log_2 k} n \log_2 n$. The leading factor for a size(UC) is twice this number, since asymptotically, the number of switches in the UC is the same as the number of nodes in $U_n^{(k)}(\Gamma_2)$, which is summarized in Table 2. We use Waksman's permutation network $P_W^{(k)}$ when calculating the size of the UC, however, even with the lower bound $P_1^{(k)}$, for $k \in \{5, 6, 7, 8\}$ we have the respective leading terms $\{4.824, 4.900, 5.190, 5\}$,

which are larger than 4.75 for $k = 4$. The last column of Table 2 shows that the smallest UC sizes are achieved in order by Zhao et al.'s optimized UC $\mathrm{UC}^{\text{Zhao et al.-4}}$, Valiant's 4-way ($k = 4$) $\mathrm{UC}^{\text{Valiant-4}}$ and 2-way UCs ($k = 2$) $\mathrm{UC}^{\text{Valiant-2}}$.

Depth. The depths of the EUG and of the UC depend only on the depth of the main skeleton, not on the subgraphs, since the longest path is between p_1 and p_n in the outest skeleton. Therefore, the asymptotic depths of EUG $U_n^{(k)}(\Gamma_1)$ and the corresponding UC are calculated as $\frac{\mathrm{depth}(B^{(k)})}{k}$, as shown in the last column of Table 3. With the lower bound $P_1^{(k)}$ for $k \in \{5, 6, 7, 8\}$ we have the respective leading terms $\{4, 4, 4.14, 4\}$, which are larger than for $k = 2$ and $k = 4$. The UC depth is minimal for Valiant's 2-way $\mathrm{UC}^{\text{Valiant-2}}$ ($k = 2$), followed by Zhao et al.'s 4-way UC $\mathrm{UC}^{\text{Zhao et al.-4}}$ and Valiant's 4-way $\mathrm{UC}^{\text{Valiant-4}}$ ($k = 4$) as shown in Table 3.

6.2 Concrete Size and Depth of UCs

In this section, we consider formulae for the concrete sizes and depths of Valiant's UCs, i.e., $\mathrm{UC}^{\text{Valiant-2}}$ and $\mathrm{UC}^{\text{Valiant-4}}$ [Val76], Zhao et al.'s method $\mathrm{UC}^{\text{Valiant-4}}$ [ZYZL18], and our hybrid universal circuits $\mathrm{UC}^{\text{H(Valiant-2,4)}}$ [GKS17] and $\mathrm{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$. Beforehand, we describe two optimizations.

6.2.1 Optimization for Fanin-1 Nodes

We observe that in $U_n^{(k)}(\Gamma_1)$ there is a fanin-1 node in the head block (cf. [GKS17, Fig. 2c and 4e] for $\mathrm{UC}^{\text{Valiant-2}}$ and $\mathrm{UC}^{\text{Valiant-4}}$, resp.). A similarly designed head block for Zhao et al.'s optimized $\mathrm{UC}^{\text{Zhao et al.-4}}$ [ZYZL18] has three such fanin-1 nodes (cf. in Fig. 18a in Appendix B). Moreover, fanin-1 nodes exist in the base-cases for a small number of poles as well [KS16]. These nodes are important to achieve fanin and fanout 2 of the graph, but can be replaced with wires when translated into a circuit description as described in §3.2. Since at least one such node can be ignored in each subgraph when nodes are translated into gates, this results in at least $k \cdot \left(\sum_{i=0}^{\log_k n-1} k^i\right) \sim kn$ less gates for the universal circuit, where $n = u + v + g$. We include this optimization in our calculations further on. This improvement decreases the depth of the UC only by a few gates.

6.2.2 Optimization for Input and Output Nodes

In the skeleton of Valiant's UC, the poles corresponding to circuit inputs need no ingoing edges and those corresponding to circuit outputs need no outgoing edges. Therefore, since u, v and g are publicly known, we optimize by deleting nodes that become redundant while cancelling the edges going to the first u (input) and coming from the last v (output) nodes. The exact number of redundant switching nodes depends on the parity or modulo 4 of u, v, $n = u + v + g$, and the k-way UC, but is $\mathcal{O}(u + v)$ in both $\Gamma_1(n)$ edge-universal graphs that build up the graph of the UC. This optimization also improves the depth by $\mathcal{O}(u + v)$.

6.2.3 Concrete Sizes and Depths of 4-Way and 2-Way UCs

We realize that based on the parity (2-way UC) and the remainder modulo 4 (4-way UC), not only the size of the outest skeleton, but also that of the smaller subgraphs can be optimized by introducing so-called head and tail blocks (cf. §3.3 and §3.4). We considered this in our 2-way UC

	Block	**Head** $H^{(k)}(\cdot)$				**Body** $B^{(k)}(\cdot)$				**Tail** $T^{(k)}(\cdot)$			
	#poles in (next) block	4	3	2	1	4	3	2	1	4	3	2	1
size	$\mathrm{UC}^{\text{Valiant-2}}$	-	-	3	-	-	-	5	5	-	-	4	1
	$\mathrm{UC}^{\text{Valiant-4}}$	13	13	12	11	19	19	18	17	14	9	4	1
	$\mathrm{UC}^{\text{Zhao et al.-4}}$	11	11	11	10	18	18	18	17	14	9	4	1
depth	$\mathrm{UC}^{\text{Valiant-2}}$	-	-	3	-	-	-	6	6	-	-	4	1
	$\mathrm{UC}^{\text{Valiant-4}}$	10	10	9	9	15	15	14	14	11	9	4	1
	$\mathrm{UC}^{\text{Zhao et al.-4}}$	9	9	9	9	14	14	14	14	11	9	4	1

Table 4: The sizes and depths of building blocks of the 2-way and 4-way UCs (cf. Figs. 3, 4a, 4b on p. 12-13, [GKS17, Figs. 2 and 4], Figs. 18a-18b in Appendix B), including the fanin-1 optimization from §6.2.1.

in [KS16], and we now generalize the approach for k-way UCs. We provide a recursive formula for the concrete size of the optimized k-way EUG as follows. Let m_k be

$$m_k := \begin{cases} n \mod k & \text{if } k \nmid n, \\ k & \text{if } k \mid n. \end{cases} \tag{13}$$

Then, given the designed head, body and tail blocks (cf. [GKS17, Figs. 2 and 4]) with sizes and depths shown in Table 4, we can compute the size by calculating the size of all the components of the outest skeleton, and the sizes of the smaller subgraphs with the recursive formula in Eq. 14.[5]

$$\begin{aligned} \mathrm{size}(U_n^{(k)}(\Gamma_1)) = \mathrm{size}(H^{(k)}(k)) + \left(\left\lceil \frac{n}{k} \right\rceil - 3\right) \cdot \mathrm{size}(B^{(k)}(k)) + \mathrm{size}(B^{(k)}(m_k)) + \mathrm{size}(T^{(k)}(m_k)) + \\ m_k \cdot \mathrm{size}\left(U^{(k)}_{\lceil \frac{n}{k}-1 \rceil}(\Gamma_1)\right) + (k - m_k) \cdot \mathrm{size}\left(U^{(k)}_{\lfloor \frac{n}{k}-1 \rfloor}(\Gamma_1)\right). \end{aligned} \tag{14}$$

As described in §3.1, a UC is constructed by means of an EUG $U_n^{(k)}(\Gamma_2)$, which is in turn constructed from two EUGs with fanin and fanout one, $U_n^{(k)}(\Gamma_1)$, by merging their poles together and thus taking them only once into consideration. When constructing a UC for circuit $C^g_{u,v}$, the number of inputs u, the number of outputs v, and the number of gates g with fanin and fanout 2 is public. Thus, using Valiant's construction, $U_n^{(k)}(\Gamma_2)$ with $n = u + v + g$ poles is constructed and thus, our formula for the concrete size of $U_n^{(k)}(\Gamma_2)$ corresponding to $C^g_{u,v}$ is

$$\mathrm{size}(U_n^{(k)}(\Gamma_2)) = 2 \cdot \mathrm{size}(U_n^{(k)}(\Gamma_1)) - n, \tag{15}$$

and the size of the UC is

$$\mathrm{size}(UC_n) \leq (\mathrm{size}(U_n^{(k)}(\Gamma_2)) - n) \cdot \mathrm{size}(X) + g \cdot \mathrm{size}(U), \tag{16}$$

where X, Y, and U denote X-, Y-switching blocks and universal gates (cf. §3.2), respectively, and $\mathrm{size}(Y) \leq \mathrm{size}(X) \leq \mathrm{size}(U)$.

[5] We note that for $k \geq 3$, there exist $H^{(k)}(k-1), \ldots, H^{(k)}(1)$ blocks. These are used for only one n, e.g., $H^{(k)}(1)$ when $n = k + 1$, and $H^{(k)}(k-1)$ when $n = 2k$. For simplicity, we consider these as special recursion base numbers in our calculations, but the formula can be adapted to include these as well.

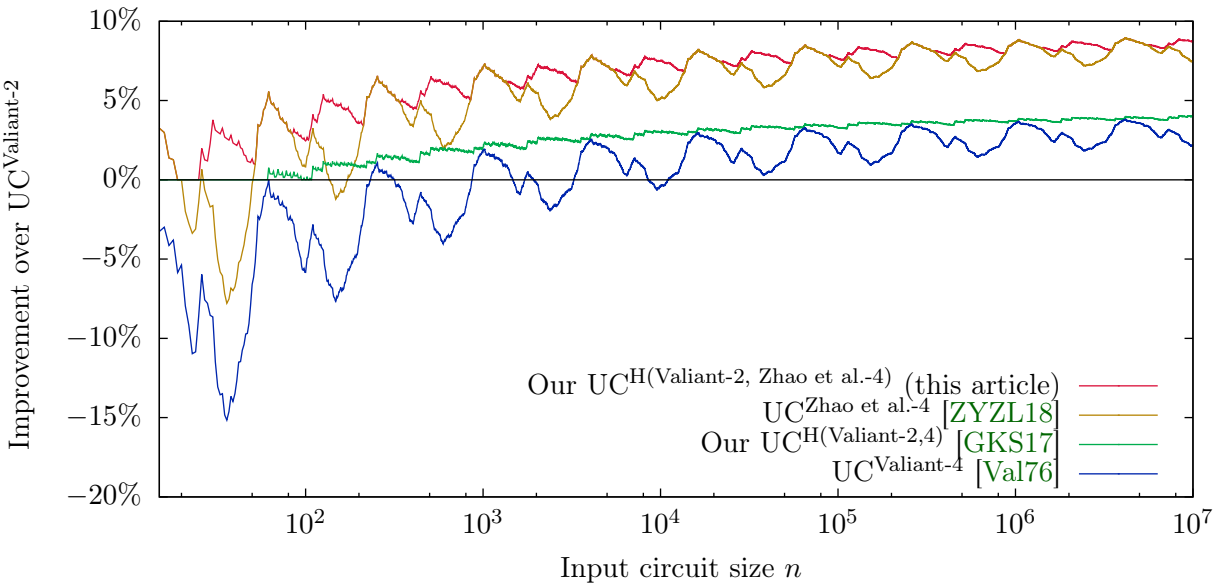


Figure 8: Improvement in size in percentage of our 2/4 hybrid, the 4-way UCs of [Val76, ZYZL18] over Valiant's 2-way UC for $15 \leq n \leq 10^7$ with logarithmic x axis.

The depth of a k-way UC also depends on m_k, the head, tail and body blocks (cf. [GKS17, Figs. 2 and 4]), but not on the subgraphs. Thus, it is calculated using the formula in Eq. 17.

$$\operatorname{depth}(U_n^{(k)}(\Gamma_1)) = \operatorname{depth}(H^{(k)}(k)) + \left(\left\lceil \frac{n}{k} \right\rceil - 3\right) \cdot \operatorname{depth}(B^{(k)}(k)) + \operatorname{depth}(B^{(k)}(m_k)) + \operatorname{depth}(T^{(k)}(m_k)). \quad (17)$$

Since $\operatorname{depth}(U_n^{(k)}(\Gamma_2)) = \operatorname{depth}(U_n^{(k)}(\Gamma_1))$, the depth of the UC is

$$\operatorname{depth}(UC_n) \leq (\operatorname{depth}(U_n^{(k)}(\Gamma_2)) - n) \cdot \operatorname{depth}(X) + g \cdot \operatorname{depth}(U), \quad (18)$$

where $\operatorname{depth}(Y) \leq \operatorname{depth}(X) \leq \operatorname{depth}(U)$.

6.2.4 Concrete Size and Depth of Our 2/4 Hybrid UC

In §5.3, we provide a construction for minimizing the concrete size of the resulting 2/4 hybrid UC. The construction chooses at each step the skeleton that results in the smallest size. We provide the recursive algorithm for determining its size in Eq. 19. Its depth is the depth of the outest skeleton, either that of the 4-way or 2-way UC, depending on which is chosen first. $\operatorname{size}(H^{(k)}(i))$, $\operatorname{size}(T^{(k)}(i))$ and $\operatorname{size}(B^{(k)}(i))$ are the values from Table 4 for $k = 2$ and $k = 4$. The size of our 2/4 hybrid UC is minimized as

$$\operatorname{size}(U_n^{\text{hybrid}(K)}(\Gamma_1)) = \min\Big(\operatorname{size}(H^{(k)}(k)) + \left(\left\lceil \frac{n}{k} \right\rceil - 3\right) \cdot \operatorname{size}(B^{(k)}(k)) + \operatorname{size}(B^{(k)}(m_k)) + \operatorname{size}(T^{(k)}(m_k)) + m_k \cdot \operatorname{size}\left(U_{\lceil \frac{n}{k} - 1 \rceil}^{\text{hybrid}(K)}(\Gamma_1)\right) + (k - m_k) \cdot \operatorname{size}\left(U_{\lfloor \frac{n}{k} - 1 \rfloor}^{\text{hybrid}(K)}(\Gamma_1)\right);\ \ k \in K = \{2, 4\}\Big), \quad (19)$$

which can be efficiently computed using a dynamic programming algorithm.

UC	Ref.	minimum	average	maximum	asymptotic
$\mathrm{UC}^{\text{Valiant-4}}$	[Val76]	-34.78%	2.97%	3.78%	5%
$\mathrm{UC}^{\text{H(Valiant-2,4)}}$	[GKS17]	0%	3.41%	4.00%	5%
$\mathrm{UC}^{\text{Zhao et al.-4}}$	[ZYZL18]	-26.09%	7.65%	8.88%	10%
$\mathrm{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$	(this article)	0%	7.71%	8.88%	10%

Table 5: Minimum, average, maximum and expected asymptotic improvement of our 2/4 hybrid and the 4-way UCs of [Val76, ZYZL18] over Valiant's 2-way UC in the range $15 \leq n \leq 10^7$.

6.2.5 Improvements over Valiant's 2-way UC

Fig. 8 shows the concrete improvement in percentage of $\mathrm{UC}^{\text{Valiant-4}}$ and $\mathrm{UC}^{\text{Zhao et al.-4}}$ over $\mathrm{UC}^{\text{Valiant-2}}$ up to ten million nodes in the simulated input circuit. All reported averages are for the interval $n \in \{15, \ldots, 10^7\}$. From the asymptotic leading factors in Table 2, we expect an improvement of up to 5% for $\mathrm{UC}^{\text{Valiant-4}}$ and 10% for $\mathrm{UC}^{\text{Zhao et al.-4}}$. In Table 5, we depict the minimum, average and maximum improvement compared to the asymptotic improvement in the interval $n \in \{2, \ldots, 10^7\}$. For the smallest n values ($n \leq 15$), $\mathrm{UC}^{\text{Valiant-2}}$ is better than both 4-way UCs. However, with growing values of n, the 4-way UCs are better, except for some short intervals as shown in Fig. 8. However, Valiant's and Zhao et al.'s 4-way UCs always outperform Valiant's 2-way UC for $n \geq 10\,996$ and $n \geq 172$, respectively, the average improvement being 2.97% and 7.65%, and the biggest improvement being 3.78% and 8.88%.

The improvement of our $\mathrm{UC}^{\text{H(Valiant-2,4)}}$ and $\mathrm{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ (cf. §5.3) is depicted in the same Fig. 8 and summarized in Table 5. For some n values our hybrid UCs achieve the same size as the 2-way or corresponding 4-way UCs, but due to their nature, their improvement is always nonnegative, and greater than or equal to the improvement achieved by the 4-way UC. Moreover, in most cases our hybrid UCs result in better sizes than the underlying 4-way UC, which means that some subgraphs are created for an n for which the 2-way UC is smaller. The overall improvement over $\mathrm{UC}^{\text{Valiant-2}}$ for all $n \in \{2, \ldots, 10^7\}$ values of our $\mathrm{UC}^{\text{H(Valiant-2,4)}}$ is on average 3.41% and at most 4.00%, and for our $\mathrm{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ is on average 7.71% and at most 8.88%.

7 Implementation and Evaluation of Our UC Compiler

In this section, we detail the challenges faced while demonstrating the practicality of Valiant's and Zhao et al.'s universal circuits. We show how to construct a UC from a standard circuit description and how to program it accordingly. We validate our results with a practical implementation that, upon receiving a fanin-2 circuit $C^g_{u,v}$ as input, outputs the corresponding 2-way or 4-way universal circuit $\mathrm{UC}^{\text{Valiant-2}}$, $\mathrm{UC}^{\text{Valiant-4}}$ or $\mathrm{UC}^{\text{Zhao et al.-4}}$ and its programming c^f. We have provided the first implementation of Valiant's 2-way UC of size $\sim 5n \log_2 n$ in [KS16]. Valiant's 4-way UC has smaller size $\sim 4.75n \log_2 n$, and we have implemented it in a modular way in [GKS17]. In this work, we extend our implementation with the modular 2-way UC and include the optimized 4-way UC with size $\sim 4.5n \log_2 n$ of Zhao et al. [ZYZL18]. We then combine the modular 2-way UC with both 4-way UCs in an implementation of our hybrid UC proposed in [GKS17] and §5.3, i.e., $\mathrm{UC}^{\text{H(Valiant-2,4)}}$ and $\mathrm{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$, respectively. Moreover, we provide a prototype implementation of our scalable 4-way UC from §5.4, which can be generalized to the 2-way UC as well as Zhao et al.'s improvement.

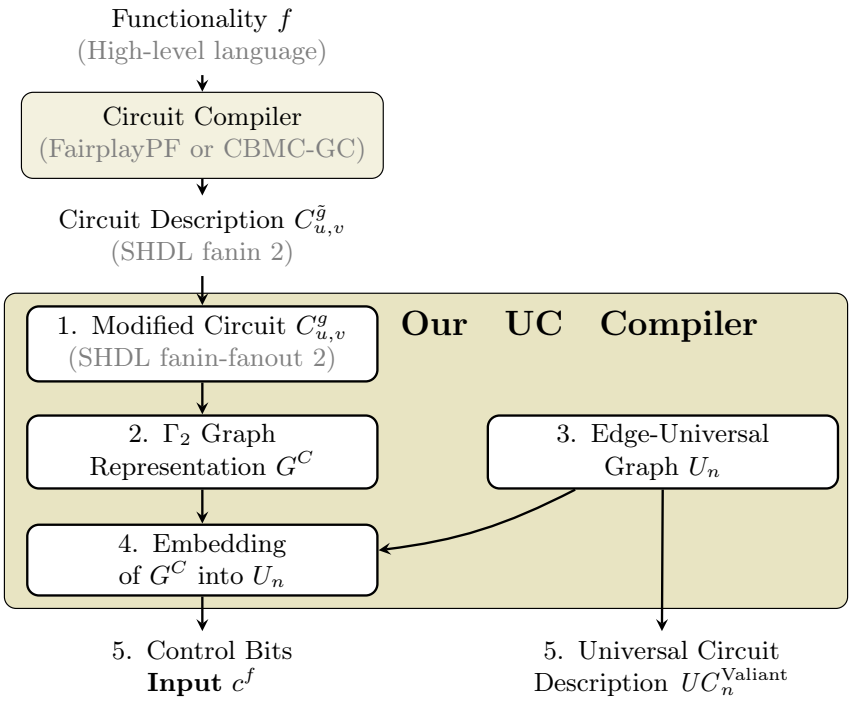


Figure 9: Our universal circuit compiler.

7.1 UC Compiler

The architecture of our UC compiler is shown in Fig. 9. In this section, we briefly describe its different artifacts and its use of the Fairplay [MNPS04] or CBMC-GC [FHK+14, BK17] frameworks as a frontend. For a more detailed description the reader is referred to [KS16]. Our implementation is available online at `https://encrypto.de/code/UC`[6].

1. Compiling Input Circuits from High-Level Functionality. We can use the Fairplay compiler [MNPS04, BNP08] with the FairplayPF extension [KS08b] or the CBMC-GC compiler [FHK+14, BK17] to translate the functionality described in a high-level language to the Fairplay circuit description called Secure Hardware Definition Language (SHDL). These compilers output a circuit $C^{\tilde{g}}_{u,v}$ with fanin 2, which is required for all UCs. However, due to Valiant's design, the input circuit $C^{g}_{u,v}$ to our UC compiler has to have fanout 2 as well, i.e., the outputs of all gates and inputs can only be used as the input of at most two subsequent gates. This can be achieved using copy gates such that instead of $\tilde{g}$ gates, we have $\tilde{g} \leq g \leq 2\tilde{g}+v$ fanout-2 gates (cf. §2.2). We give concrete examples in [KS16] on how this conversion affects the size of practical circuits and show that in most cases, the resulting number of gates remains significantly below the upper bound $2\tilde{g} + v$.

2. Obtaining the $\Gamma_2(n)$ Graph G of the Circuit $C^{g}_{u,v}$. As next step, we transform circuit $C^{g}_{u,v}$ into a $\Gamma_2(n)$ graph $G = (V, E)$ with $n = u + v + g$ (cf. §3.1). This can directly be generated as described in §2.2: with the number of inputs u, outputs v and gates g in circuit $C^{g}_{u,v}$, G has n nodes

[6] We will release the code for the optimized block by [ZYZL18] $\text{UC}^{\text{Zhao et al.-4}}$ and our hybrid UC constructions $\text{UC}^{\text{H(Valiant-2,4)}}$ and $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ soon.

and the wires are represented as edges in the graph. Then, we define a topological order η^G on the nodes of G such that every input node v_i has a topological order of $1 \leq \eta^G(v_i) \leq u$ and every output node v_j is labelled with $u+g+1 \leq \eta^G(v_j) \leq u+v+g$. Since $C^g_{u,v}$ has fanin and fanout 2, the resulting graph G is in $\Gamma_2(n)$, where $n = u+v+g$.

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. Therefore, 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 UC in Step 5.

3. Generating Edge-Universal Graph $U_n^{(\ell)}(\Gamma_2)$ or $U_n^{\textbf{hybrid}(K)}(\Gamma_2)$ for $\Gamma_2(n)$ graphs, where $\ell \in \{2,4\}$ and $K = \{2,4\}$. An EUG $U_n^{(\ell)}(\Gamma_2)$ or $U_n^{\text{hybrid}(K)}(\Gamma_2)$ is constructed by creating two instances of $U_n^{(\ell)}(\Gamma_1)$ or $U_n^{\text{hybrid}(K)}(\Gamma_1)$, resp., as described in §3.1. The two instances get merged to $U_n^{(\ell)}(\Gamma_2)$ so that one builds the left inputs and outputs and the other builds the right inputs and outputs of the gates (based on the two-coloring of G). For efficiency reasons, we directly generate the merged edge-universal graph, i.e., an EUG for $\Gamma_2(n)$, with the poles as common nodes. We partly include our optimization for the input and output nodes from §6.2.2[7] and Valiant's optimizations for the base cases $n \in \{2,3,4\}$, but do not consider Valiant's optimizations for $n \in \{5,6\}$ [Val76]. Knowing the number of input bits u, the number of gates g and the number of output bits v, we construct the corresponding edge-universal graph $U_n^{\ell}(\Gamma_2)$, where $n = u+v+g$. We note that no knowledge is necessary about the topology or the gate tables in circuit C for this step.

4. Programming $U_n^{\ell}(\Gamma_2)$ and $U_n^{\textbf{hybrid}(K)}(\Gamma_2)$ According to an Arbitrary $\Gamma_2(n)$ Graph. We edge-embed graph G into $U_n^{(\ell)}(\Gamma_2)$ as described in §4 and into our hybrid $U_n^{\text{hybrid}(K)}(\Gamma_2)$ with $K = \{2,4\}$ as described in §5.3. G is partitioned into two $\Gamma_1(n)$ graphs G_1 and G_2 which are embedded into the two EUGs $U_n^{\ell}(\Gamma_1)_1$ and $U_n^{\ell}(\Gamma_1)_2$. Valiant proved in [Val76] that any topologically ordered $\Gamma_1(n)$ graph can be edge-embedded in an EUG $U_n^{\ell}(\Gamma_1)$ (cf. §3.1). We perform the embedding as described in §4 for Valiant's 2-way and 4-way EUGs in Listing 1. The difference when using Zhao et al.'s improvement [ZYZL18] is the block edge-embedding described in §4.1. Here, we utilize a lookup table derived from the computer generated proof of Zhao et al. [ZYZL18] that maps the *in* and *out* vectors as defined in §4.1 into the programming bits of the block, i.e., can be used as block edge-embedding along with the recursion point edge-embedding described in §4.2. We edge-embed G_1 and G_2 into our 2/4-hybrid EUGs $U_n^{\text{hybrid}(K)}(\Gamma_1)_1$ and $U_n^{\text{hybrid}(K)}(\Gamma_1)_2$ as described in §5.3. When the edge-embedding is finished, we define the control bits of the programmable blocks (universal gates and switches) as described in §3.2.

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 $U_n(\Gamma_2)$, we write the resulting circuit in a file using our generic UC description. In the edge-universal graph, each node stores the control bit resulting from the edge-embedding (control bit c of the corresponding universal switch in §3.2) and each pole corresponding to a gate stores four bits (the four control bits of the function table of the corresponding gate in the original circuit $C^g_{u,v}$, c_0, c_1, c_2, c_3 in Eq. 1,

[7] We delete edges coming into inputs and going out from outputs. Due to this, some nodes are removed due to our fanin-1 optimization §6.2.1 when translated into a UC.

their order possibly changed in Step 2). Thus, after topologically ordering $U_n(\Gamma_2)$, one can directly write out the gate identifiers into a circuit file UC and the control bits to a programming file c^f. We include our optimization from §6.2.1, and ignore extra nodes with fanin 1 when the graph is translated into a UC description. This improves the size of the recursion bases for $n = \{4, 5, 6\}$ as well as of the head blocks [GKS17, Fig. 2c and Fig, 4e] and Fig. 18a in Appendix B.

Our circuit description format is generic, i.e., consists of universal switches and universal gates. Therefore, any framework can be adapted to use them, independently from if it is interpreted as a Boolean or arithmetic UC. We start with enumerating the client input wires as $C \quad 0 \quad 1 \quad \ldots \quad u-1$. As a reminder, the $\mathcal{O}(n \log n)$ server input wires are in the programming file c^f. In the UC, 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):

$$U \quad \text{in}_1 \quad \text{in}_2 \quad \text{out}_1 \tag{20}$$

$$X \quad \text{in}_1 \quad \text{in}_2 \quad \text{out}_1 \quad \text{out}_2 \tag{21}$$

$$Y \quad \text{in}_1 \quad \text{in}_2 \quad \text{out}_1 \tag{22}$$

denotes that wire out_1 (and possibly out_2) is coming from a gate with input wires in_1 and in_2. The control bits are not represented in the circuit format, but for each universal gate we save a four-bit number representing the control bits and for each universal switch we store the control bit in the programming file c^f. 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 as shown in Listings 3 and 4 in Appendix C.

7.2 Experimental Evaluation

We ran all experiments for our UC compiler on a Desktop PC, equipped with an Intel Core i7-4790 CPU with 3.6 GHz and 32 GB RAM, and provide our results in this section. We performed experiments for circuit sizes $n \in \{10, 100, \ldots, 1\,000\,000\}$ as well as with notable circuits from [TS18] such as the AES-128 circuit without key expansion with size $n = 38\,518$ and the SHA-256 circuit with size $n = 201\,206$. We note that these sizes are for the circuits transformed to have fanin and fanout 2 as described in [KS16, Table 1].

Circuit Sizes (Fig. 10). We first compare the circuit sizes of our implementations that slightly differ from the expected sizes shown in §6. Our initial 2-way $\text{UC}^{\text{Valiant-2}}$ implementation from [KS16] included the recursion bases for 1, 2, and 3 nodes, however, did not include those proposed by Valiant [Val76] optimized for 4, 5, and 6 nodes. It included both size optimizations described in §6.2.1 and §6.2.2. In Fig. 10 we show the improvement over our $\text{UC}^{\text{Valiant-2}}$ implementation from [KS16] in percentage of the number of switches of our later, more modular UC implementations presented in this article and in [GKS17]. We note that the number of universal gates is the same for all implementations, i.e., the number of gates in the original circuits g.

Our modular 4-way $\text{UC}^{\text{Valiant-4}}$ implementation from [GKS17] additionally included the recursion base with 4 nodes, however, only partly included the optimization described in §6.2.2 concerning the input and output nodes. The edges directed into the inputs and out of the outputs are also removed which results in smaller sizes due to the thus redundant nodes, however, not all unnecessary connections are deleted. This, however, incurs only a small overhead of at most $\mathcal{O}(u + v)$. As we

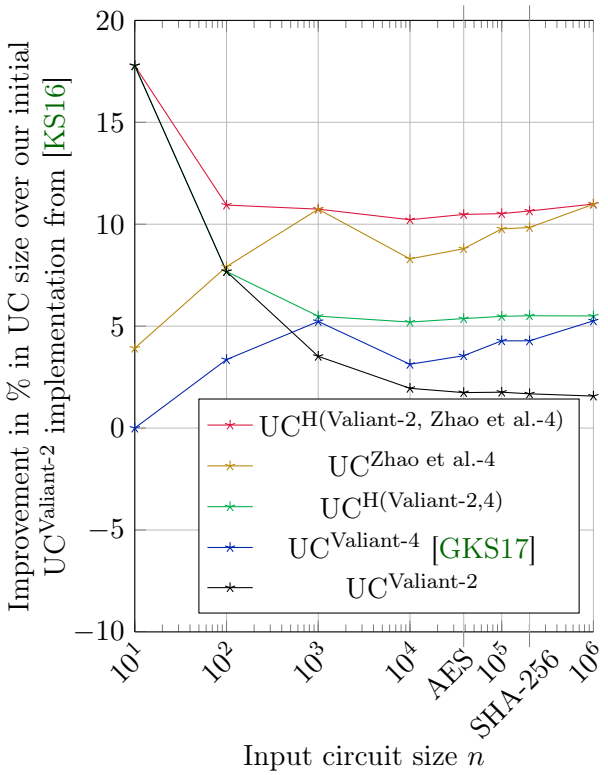


Figure 10: Improvement in percentage of the UC sizes (number of switches) of our UC implementation of Valiant's 4-way $\mathrm{UC}^{\text{Valiant-4}}$ from [GKS17] and our novel implementations including a modular version of Valiant's 2-way $\mathrm{UC}^{\text{Valiant-2}}$, Zhao et al.'s improved block $\mathrm{UC}^{\text{Zhao et al.-4}}$ and hybrid constructions $\mathrm{UC}^{\text{H(Valiant-2,4)}}$ and $\mathrm{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ over our implementation of Valiant's 2-way UC from [KS16].

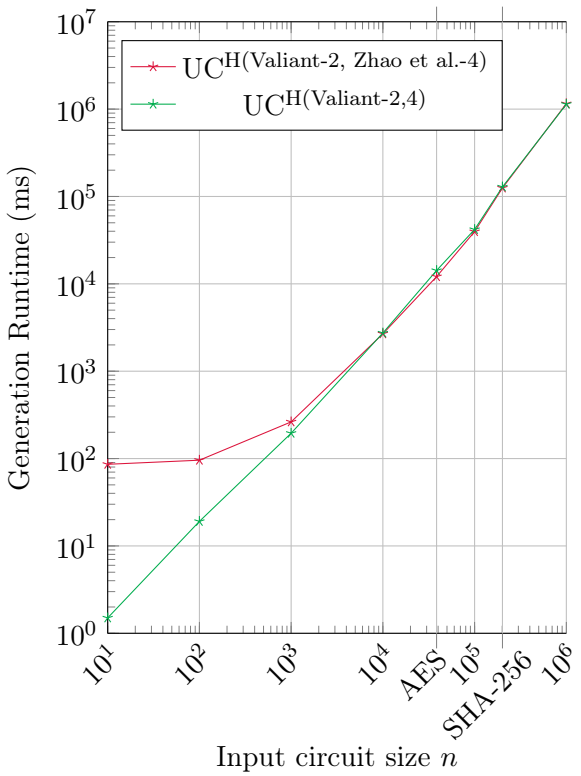


Figure 11: Comparison of the runtime of our hybrid UC implementations using either Valiant's 2-way and 4-way UCs or Valiant's 2-way UC with Zhao et al.'s improved block. We note that the runtime of $\mathrm{UC}^{\text{Zhao et al.-4}}$ only slightly differs from that of $\mathrm{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$, and the runtimes of $\mathrm{UC}^{\text{Valiant-2}}$ and $\mathrm{UC}^{\text{Valiant-4}}$ only slightly differ from that of $\mathrm{UC}^{\text{H(Valiant-2,4)}}$, and therefore we omit them from the figure.

can observe in Fig. 10 and as expected (cf. Table 5 on p. 31), this implementation improved by around 5% over our implementation from [KS16].

In this article, we have first implemented the modular version of Valiant's 2-way $\mathrm{UC}^{\text{Valiant-2}}$ where inherently we use the optimized recursion base with 4 nodes as well. An around 1.5-2% improvement can be observed over our non-modular implementation from [KS16]. Using this and our modular 4-way $\mathrm{UC}^{\text{Valiant-4}}$, we have implemented our hybrid $\mathrm{UC}^{\text{H(Valiant-2,4)}}$ using Valiant's 2-way and 4-way UC as proposed in [GKS17]. This implementation has a more steady improvement of at least 5% for most tested circuit sizes. Moreover, we also implemented the optimized $\mathrm{UC}^{\text{Zhao et al.-4}}$ proposed in [ZYZL18], who have proved that their optimized block is universal by giving the programming for all possible path combinations in the block. We use this proof to generate a lookup table file for our implementation, that contains a mapping from any possible input-output vector (cf. §4.1) and the corresponding programming bits for the block. The generation of this lookup

Input circuit size n	10	100	1 000	10 000	38 518 (AES)	100 000	201 206 (SHA-256)	1 000 000
Size (#switches)	45	1 719	31 667	462 667	2 119 836	6 147 387	13 277 772	76 484 267
UC file (KB)	0.6	36	794	13 473	68 730	207 789	473 915	2 936 852
Prog. file c^f (KB)	0.1	4	65	933	4 224	12 300	26 391	152 314

Table 6: Size of our smallest UCs generated with $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$, i.e., its number of switches, the sizes of the UC and programming files.

table is a one-time precomputation cost and takes around 82 seconds. In subsequent runs of the UC compiler, this overhead is no longer needed and a file of size 1.08 MB is read which takes only about 80 milliseconds. Thereafter, the expected gain of around 10% can be observed over our 2-way $\text{UC}^{\text{Valiant-2}}$ implementation from [KS16]. Moreover, the hybrid variant with this construction, i.e., $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ achieves an at least 10% improvement for all our example circuits.

In Table 6 we show the concrete number of switches of the smallest UCs generated with $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ as well as the sizes of the resulting UC and programming files. The universal circuit for $n = 1$ million gates has around 76 million switches and additionally around 1 million universal gates (which, in the PFE setting results in a total of about 77 million AND gates for Yao's garbled circuit protocol and 79 million AND gates for the GMW protocol). The corresponding file for the UC has size 2.8 GB and the programming file has size 0.15 GB.

Runtime (Fig. 11). To compare the runtime of our UC implementation with that of the UC compiler of [KS16], we ran the same experiments on the same platform using our novel implementations for $\text{UC}^{\text{Valiant-2}}$, $\text{UC}^{\text{Zhao et al.-4}}$, $\text{UC}^{\text{H(Valiant-2,4)}}$ and $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$. Runtimes are reported as averages from 10 executions. The differences in runtimes for the different constructions are not significant, and therefore, we only depict the runtimes of our hybrid implementations $\text{UC}^{\text{H(Valiant-2,4)}}$ and $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ in Fig. 11.

The runtimes of our modular $\text{UC}^{\text{Valiant-2}}$ and $\text{UC}^{\text{Valiant-4}}$ implementations are very similar to those of $\text{UC}^{\text{H(Valiant-2,4)}}$, the latter of which becomes best for larger circuits (i.e., our examples with $n \geq 10\,000$). The runtimes of $\text{UC}^{\text{Zhao et al.-4}}$ are only slightly lower than those of our hybrid $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$, both of which include a one-time overhead of around 80 milliseconds for reading in our lookup table of size 1.08 MB for each possible block programming [ZYZL18]. However, this one-time expense is only significant for small circuits as can be observed in Fig. 11, and $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ becomes faster than $\text{UC}^{\text{H(Valiant-2,4)}}$ for our examples with $n \geq 10\,000$. The runtime of our original 2-way $\text{UC}^{\text{Valiant-2}}$ from [KS16] was slightly better due to its handling of the UC as one big block. However, it also becomes worse than $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ for our largest examples SHA-256 and the circuit for one million gates due to the gain in the size that results in a less complex embedding. For instance, it takes about 12 seconds to generate the smallest $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ with our new implementation for AES-128, while our original implementation for $\text{UC}^{\text{Valiant-2}}$ took 9.4 seconds. Our largest examples SHA-256 and a circuit with one million gates were generated and programmed in 2.1 and 18.6 minutes, respectively. The runtimes are high for these large examples, however, they are generally a one-time precomputation expense in most application scenarios such as private function evaluation (cf. §1.1).

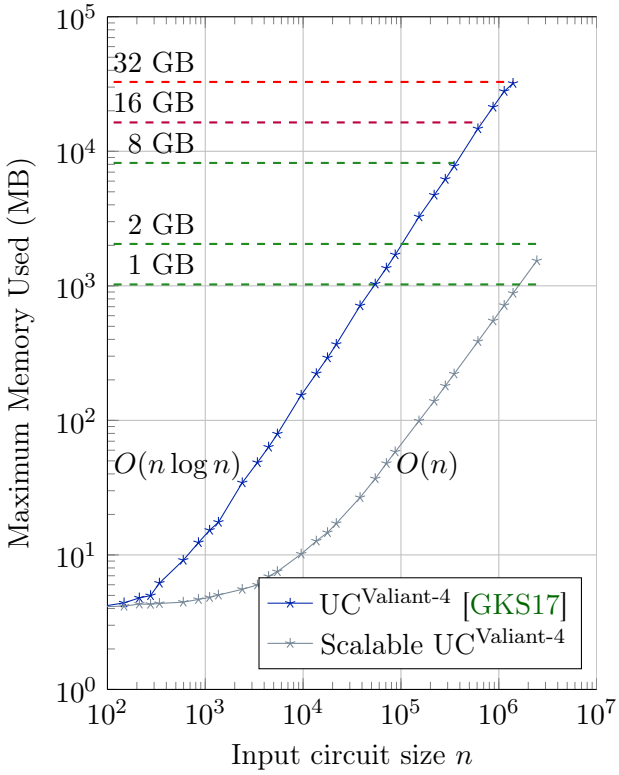


Figure 12: Comparison of the maximum memory used between our per-block and [GKS17]'s UC generation. [GKS17]'s implementation runs out of 32 GB of memory for $n > 1\,398\,100$ nodes.

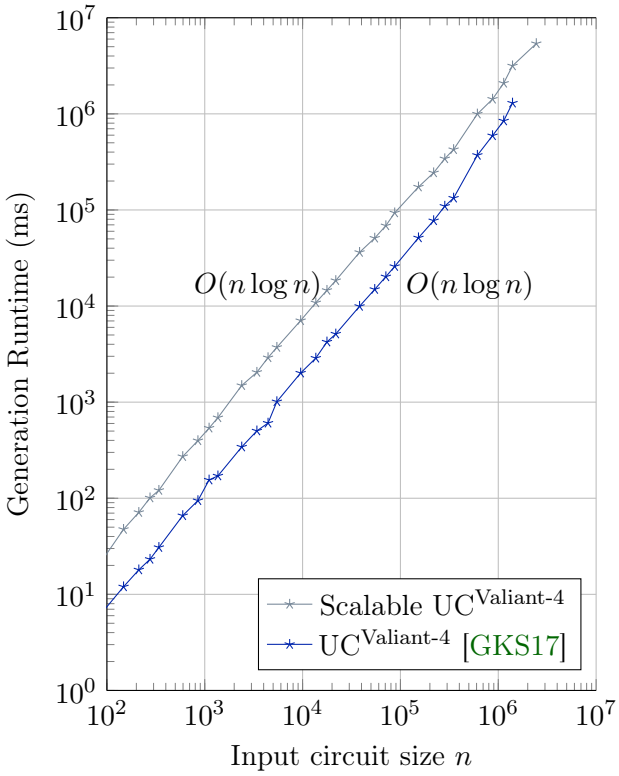


Figure 13: Comparison of the runtime of our per-block and [GKS17]'s UC generations for up to about $n = 2\,446\,000$ nodes, which fails with [GKS17]'s UC generation and 32 GB of memory.

Scalable 4-Way UC Implementation (Figs. 12-13). We also implemented our scalable 4-way UC generation algorithm presented in §5.4. We note that our implementation only includes H^i, T^i_x and B^i_x for $i = 0, 1, 2, 3$ and $x = 4$, and does not include the optimized versions for $x = 1, 2, 3$ which we leave as future work. Moreover, we include the base cases for $n = 1, 2, 3$ but not that for $n = 4$. This is due to the fact that a lot of engineering effort would be required for including the other options as well and our work is only a proof-of-concept implementation of our method presented in §5.4. Therefore, we test circuits with specific sizes where none of the other blocks or base case are required, i.e., where all subgraphs at each recursion step have 4 nodes in the tail block and the base case with $n = 4$ is not needed. Currently, for generating UCs for different sizes, one would need to pad the original circuit with dummy gates to an allowed size. Our aim was to improve the memory consumption of the UC generation (and programming) algorithm, while keeping the price paid in runtime as low as possible. The number of files created is the number of subgraphs in the UC, which is necessary for efficient scalable programming of the UC.

We show that our scalable UC generation implementation provides the expected improvement in memory usage by comparing our scalable UC$^{\text{Valiant-4}}$ implementation to our implementation from [GKS17]. We depict in Fig. 12 the memory usage of the generation algorithm with growing input circuit sizes on a machine with 32 GB RAM memory. As can be seen in the figure, instead

of holding the whole UC of size $\mathcal{O}(n \log n)$ in memory, we indeed hold only $\mathcal{O}(n)$ information in memory at each step. When using 1 GB, 8 GB, and 32 GB of memory, we can generate a UC for over 27×, 28×, and 29× larger input circuit sizes n, respectively. Moreover, as can be observed in Fig. 13, the runtime of the resulting scalable UC generation is only around 4× that of the $\text{UC}^{\text{Valiant-4}}$ implementation of [GKS17]. This difference is becoming smaller with increasing n due to the fact that the implementation of [GKS17] is running short on memory and starts swapping to disk. Our experiments show that while reducing the memory requirements of our UC generation for $\text{UC}^{\text{Valiant-4}}$, we keep the runtime asymptotically the same (cf. Fig. 13). Moreover, the required storage capacity is also $\mathcal{O}(n \log n)$ as before, since the additionally stored data at each step is at most $\mathcal{O}(n)$, cf. §5.4.

8 Toolchain for Private Function Evaluation

Secure function evaluation (SFE) allows two parties to jointly compute a public function on their private inputs, without revealing anything to each other besides the output of the computation. As it is probably the most prominent application of UCs (cf. §1.1), we implement private function evaluation (PFE) using SFE of a Boolean universal circuit. In this scenario, one of the parties holds its input x and the other party holds the programming c^f corresponding to a private function f that allows the UC to compute $UC(x, c^f) = f(x)$. We note that the UC (with control bits for the universal gates and switches) can be publicly generated.

We create a novel toolchain for private function evaluation (PFE), using the ABY framework for SFE as backend of our UC compiler. ABY implements state-of-the-art optimizations of Yao's garbled circuit protocol [Yao82, Yao86] and the GMW protocol [GMW87]. 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 §1.1.

8.1 Extension of the ABY Framework

We adapt the ABY secure two-party computation framework [DSZ15] for securely evaluating universal circuits. We realize the universal circuit building blocks (universal gates and switches) with a number of AND and XOR gates, which is the functionally complete set of logical gates that ABY uses. Since XOR gates can be evaluated for free in the underlying protocols for secure function evaluation due to the free-XOR optimization [KS08a], from here on, we study the *AND-size* (size^{AND}) and *AND-depth* ($\text{depth}^{\text{AND}}$) of UCs, i.e., the number of AND gates and the maximum number of AND gates on the longest path, respectively. For other applications, however, the *total sizes and depths* of the UCs with respect to both AND and XOR gates are relevant. We implement universal gates and switches optimized for PFE and therefore use few AND gates, and only (free) XOR gates alongside it. X and Y gates are obtained as shown in [KS08a]

$$\text{out}_1 = Y(\text{in}_1, \text{in}_2; c) = (\text{in}_1 \oplus \text{in}_2)c \oplus \text{in}_1 \tag{23}$$

$$(\text{out}_1, \text{out}_2) = X(\text{in}_1, \text{in}_2; c) = (e \oplus \text{in}_1, e \oplus \text{in}_2) \text{ with } e = (\text{in}_1 \oplus \text{in}_2)c \tag{24}$$

with $\text{size}^{\text{AND}}(Y) = \text{size}^{\text{AND}}(X) = \text{depth}^{\text{AND}}(Y) = \text{depth}^{\text{AND}}(X) = 1$ for both universal switches. In case the SFE implementation uses Yao's garbled circuit protocol [Yao86], both $\text{size}^{\text{AND}}(U) = 1$ and $\text{depth}^{\text{AND}}(U) = 1$, due to the fact that in some garbling schemes the evaluator does not learn the type of the evaluated gate such as in the case of garbled 3-row-reduction (GRR3) [NPS99]. Therefore,

a universal gate can be implemented using only one 2-input non-XOR gate [PSS09]. For other SFE protocols such as GMW where this optimization is not possible, our efficient implementation of generic universal gates uses Y gates yielding

$$\text{out}_1 = U(\text{in}_1, \text{in}_2; c_0, c_1, c_2, c_3) = Y[Y(c_0, c_1; \text{in}_2), Y(c_2, c_3; \text{in}_2); \text{in}_1] \tag{25}$$

with $\text{size}^{\text{AND}}(U) = 3$ and $\text{depth}^{\text{AND}}(U) = 2$. We note that the implementation of switches and universal gates might look very different when other 2-input Boolean gates can also be used, e.g., when other size metrics are to be minimized.

We include our implementation of these efficient UC building blocks in the open-source ABY framework `https://encrypto.de/code/ABY`. For evaluating a UC securely, the output universal circuit file of our UC compiler is parsed, a circuit UC is generated and evaluated with the input x and the control bits c^f to compute $f(x)$. Our toolchain is the first implementation of Valiant's size-optimized UC that supports efficient private function evaluation [KS16].

8.2 Experimental Results

We validate the practicality of our implementation, which is the first practical implementation of private function evaluation (PFE), cf. §1.1. We ran our experiments on two Desktop PCs, each equipped with an Intel Core i9-7960X CPU with 2.8 GHz and 128 GB RAM. We give the runtimes in Fig. 14 and communication in Fig. 15 for our example circuits from the previous section, i.e., for random circuits of sizes $n \in \{10, 100, \ldots, 1\,000\,000\}$ as well as the AES and SHA-256 circuits from [TS18]. For completeness, we give the exact numbers in Table 7 in Appendix D. Our runtime measurements are provided from an average of 10 executions, in two different settings: in a LAN setting with 10 Gbit/s bandwidth and 1ms RTT, as well as in a simulated WAN setting with 100 Mbit/s bandwidth and 100ms RTT.

We evaluate UCs in ABY [DSZ15] both with the GMW protocol [GMW87] and Yao's garbled circuit protocol [Yao82] with state-of-the-art optimizations. Yao's garbled circuit protocol achieves much better runtimes than that of the GMW protocol since the latter has $\mathcal{O}(n)$ rounds (i.e., the number of rounds is the depth of the circuit, and Valiant's UCs have depth $\mathcal{O}(n)$, cf. §6.1 and Table 7 in Appendix D) whereas Yao's protocol runs in 3 rounds. The effect of this is especially apparent in the WAN setting where the round-trip time is much higher. In both settings, the runtime of the GMW protocol is dominated by the linear term due to the linear number of online rounds. The amount of communication is similar in both implementations, however, it could be reduced by half for Yao's protocol if X and Y switches would be implemented with the optimization from [KS08a] using only one ciphertext. The current implementation utilizes two ciphertexts per X and Y switches.

Due to the clear advantage of Yao's protocol over the GMW protocol, we highly recommend using Yao's protocol when evaluating UCs securely for PFE. Investigating depth-optimized UCs [CH85] with $\mathcal{O}(d)$ depth in the depth of the input circuit d could improve the performance of the GMW protocol, however, its number of rounds will still depend on d whereas Yao's protocol runs in only 3 rounds.

8.3 Comparison of PFE Approaches

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 garbled circuits [Yao86], and to

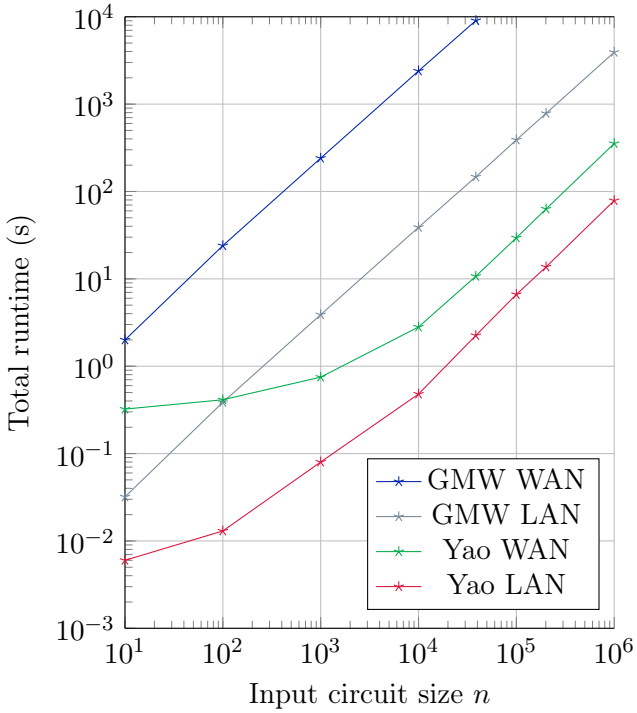


Figure 14: Total runtime in seconds on LAN/WAN of PFE with the best available UC variant $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$.

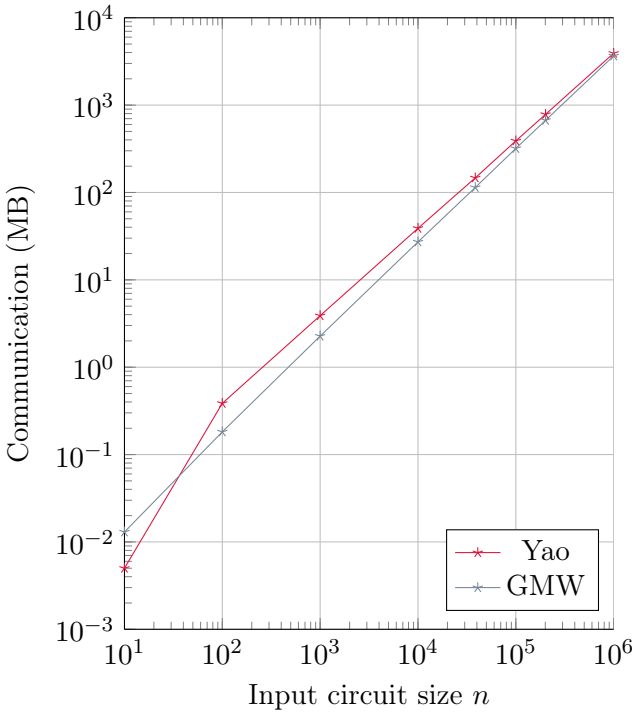


Figure 15: Total communication in megabytes of PFE with the best available UC variant $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$.

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

As 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 $4\tilde{g}\log_2(2\tilde{g})+1$ that are evaluated using OT extension, for which they calculate $8\tilde{g}\log_2(2\tilde{g})+8$ symmetric-key operations together with $5\tilde{g}$ 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., $4\tilde{g}$ symmetric-key operations for creating a garbled circuit with $\tilde{g}$ 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 $12\tilde{g}\log_2(2\tilde{g})+4\tilde{g}+12$ symmetric-key operations.

In the same way, we assume that in the case of PFE with UCs, for both the universal gates and switches, the garbler needs $4n$ symmetric-key operations. In this case, however, $n = u+v+g$, where

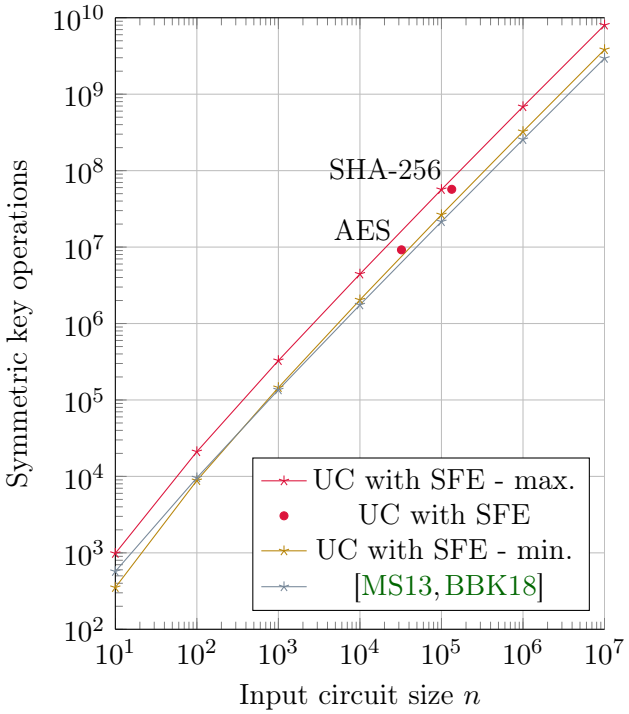


Figure 16: The number of symmetric-key operations of different PFE protocols: Valiant's UC with Yao's garbled circuits, Mohassel et al.'s OT-based method from [MS13] and its optimized version from [BBK18].

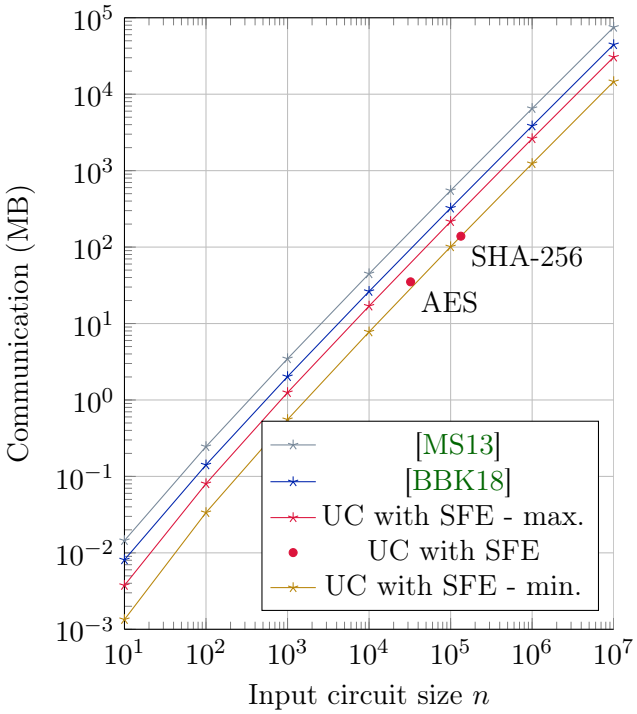


Figure 17: Communication of different PFE protocols in megabytes: $\mathrm{UC}^{\mathrm{H(Valiant\text{-}2,\ Zhao\ et\ al.\text{-}4)}}$ with Yao's garbled circuits, Mohassel et al.'s OT-based method from [MS13] and its optimized version from [BBK18].

$\tilde{g} \leq g \leq 2\tilde{g} + v$. It is, therefore, difficult to directly compare complexities of specifically designed protocols with $\tilde{g}$ fanin-2 gates and UCs where the input circuit is required to have fanout 2 as well. In Fig. 16, we therefore depict the minimum and maximum required number of symmetric-key operations for circuits with size $\tilde{g} \in \{10, 100, \ldots, 1\,000\,000\}$. Moreover, we depict the concrete values with real-world circuits (AES-128 and SHA-256 from [TS18]) with UC with SFE, and note that for the other approaches the points lie on the corresponding line.

The protocol of [MS13] has been improved to achieve better communication in [BBK18]. The communication of the protocol of [MS13] is $(10\tilde{g}\log_2\tilde{g} + 4\tilde{g} + 5) \cdot 128$, while that of [BBK18] is $(6\tilde{g}\log_2\tilde{g} + 0.5\tilde{g} + 3) \cdot 128$. For SFE with UC, we require one ciphertext per X and Y switches [KS08a] and $3 \cdot 2$ ciphertexts per universal gates. Fig. 17 depicts the comparison between the communication of SFE with UCs with minimum and maximum values depending on the relation of g and $\tilde{g}$ as before and the alternatives of [MS13] and [BBK18]. We can see that SFE with UCs always achieves the best communication, requiring 1.5-3× less communication than the improvement of [BBK18].

9 Conclusion

Universal circuits (UCs) are highly relevant for various applications such as verifiable computation, attribute-based encryption, and private function evaluation (PFE) which can for example be used for

privacy-preserving evaluation of diagnostic programs, proprietary software and in private database management systems. These applications require size-optimized universal circuits, first proposed by Valiant [Val76]. Since then, several optimizations appeared to further reduce the size of the UCs.

In this article, we revisit Valiant's original constructions and the optimizations later proposed by our previous works by Kiss and Schneider [KS16] and Günther et al. [GKS17] as well as by Zhao et al. [ZYZL18]. We have shown the practicality of Valiant's universal circuit constructions and its several improvements by providing the implementation of the most efficient UC to date with size $\sim 4.5n \log_2 n$ in the input circuit size n. Moreover, we highly improve the memory consumption of our UC generation algorithm by designing and implementing a method that utilizes $\mathcal{O}(n)$ memory instead of the previous methods using $\mathcal{O}(n \log n)$ memory.

Universal circuits for an input circuit size of one million can be generated and programmed within a matter of around 18 minutes on a standard PC and utilized in various applications. We demonstrate the practicality of PFE with the secure evaluation of UCs and show that such a large universal circuit can be evaluated within 1.3 and 5.9 minutes in LAN and WAN setting, respectively using Yao's garbled circuit protocol.

Acknowledgements

This work was supported by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within the National Research Center for Applied Cybersecurity CRISP, by the DFG as part of project E4 within the CRC 1119 CROSSING, and by the European Union's 7th Framework Program (FP7/2007-2013) under grant agreement n. 609611 (PRACTICE). We thank Michael Zohner for helping with the implementation in ABY and the anonymous reviewers of EUROCRYPT'16 and ASIACRYPT'17 for their helpful comments.

References

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

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS'13*, pages 535–548. ACM, 2013.

[AMPR14] A. Afshar, P. Mohassel, B. Pinkas, and B. Riva. Non-interactive secure computation based on cut-and-choose. In *EUROCRYPT'14*, volume 8441 of *LNCS*, pages 387–404. Springer, 2014.

[Att14] N. Attrapadung. Fully secure and succinct attribute based encryption for circuits from multi-linear maps. Cryptology ePrint Archive, Report 2014/772, 2014. https://ia.cr/2014/772.

[BBK18] O. Biçer, M. A. Bingöl, and M. S. Kiraz. Highly efficient and reusable private function evaluation with linear complexity. *Cryptology ePrint Archive, Report 2018/515*, 2018. https://ia.cr/2018/515.

[BBKL17] O. Biçer, M. A. Bingöl, M. S. Kiraz, and A. Levi. Towards practical PFE: An efficient 2-party private function evaluation protocol based on half gates. Cryptology ePrint Archive, Report 2017/415, 2017. https://ia.cr/2017/415.

[BD02] B. Beauquier and É. Darrot. On arbitrary size Waksman networks and their vulnerability. *Parallel Processing Letters*, 12(3-4):287–296, 2002.

[BFGH10] D. Bera, S. A. Fenner, F. Green, and S. Homer. Efficient universal quantum circuits. *Quantum Information and Computation*, 10(1–2):16–27, 2010.

[BFK+09] 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 *ESORICS'09*, volume 5789 of *LNCS*, pages 424–439. Springer, 2009.

[BK17] Niklas Büscher and Stefan Katzenbeisser. *Compilation for Secure Multi-party Computation*. Springer Briefs in Computer Science. Springer, 2017.

[BNP08] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: A system for secure multi-party computation. In *CCS'08*, pages 257–266. ACM, 2008.

[BOKP15] S. Banescu, M. Ochoa, N. Kunze, and A. Pretschner. Idea: Benchmarking indistinguishability obfuscation - A candidate implementation. In *Engineering Secure Software and Systems (ESSoS'15)*, volume 8978 of *LNCS*, pages 149–156. Springer, 2015.

[BPSW07] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *CCS'07*, pages 498–507. ACM, 2007.

[BV15] N. Bitansky and V. Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS'15*, pages 171–190. IEEE, 2015.

[CCKM00] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *ICALP'00*, volume 1853 of *LNCS*, pages 512–523. Springer, 2000.

[CDN01] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT'01*, volume 2045 of *LNCS*, pages 280–299. Springer, 2001.

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

[DDKZ13] K. Durnoga, S. Dziembowski, T. Kazana, and M. Zajac. One-time programs with limited memory. In *Information Security and Cryptology (INSCRYPT'13)*, volume 8567 of *LNCS*, pages 377–394. Springer, 2013.

[DSZ15] D. Demmler, T. Schneider, and M. Zohner. ABY – a framework for efficient mixed-protocol secure two-party computation. In *NDSS'15*. The Internet Society, 2015. Code: https://encrypto.de/code/ABY.

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

[FAZ05] K. B. Frikken, M. J. Atallah, and C. Zhang. Privacy-preserving credit checking. In *Electronic Commerce (EC'05)*, pages 147–154. ACM, 2005.

[FGP14] D. Fiore, R. Gennaro, and V. Pastro. Efficiently verifiable computation on encrypted data. In *CCS'15*, pages 844–855. ACM, 2014.

[FHK+14] Martin Franz, Andreas Holzer, Stefan Katzenbeisser, Christian Schallhart, and Helmut Veith. CBMC-GC: an ANSI C compiler for secure two-party computations. In *Conference on Compiler Construction (CC'14)*, volume 8409 of *LNCS*, pages 244–249. Springer, 2014.

[FLA06] K. B. Frikken, J. Li, and M. J. Atallah. Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *NDSS'06*, pages 157–172. The Internet Society, 2006.

[FVK+15] B. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin. Malicious-client security in Blind Seer: A scalable private DBMS. In *IEEE S&P'15*, pages 395–410. IEEE, 2015.

[GGH+13a] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS'13*, pages 40–49. IEEE, 2013.

[GGH+13b] S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO'13*, volume 8043 of *LNCS*, pages 479–499. Springer, 2013.

[GGHZ14] S. Garg, C. Gentry, S. Halevi, and M. Zhandry. Fully secure attribute based encryption from multilinear maps. Cryptology ePrint Archive, Report 2014/622, 2014. https://ia.cr/2014/622.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT'13*, volume 7881 of *LNCS*, pages 626–645. Springer, 2013.

[GHV10] C. Gentry, S. Halevi, and V. Vaikuntanathan. i-hop homomorphic encryption and rerandomizable Yao circuits. In *CRYPTO'10*, volume 6223 of *LNCS*, pages 155–172. Springer, 2010.

[GKS17] Daniel Günther, Á. Kiss, and T. Schneider. More efficient universal circuit constructions. In *ASIACRYPT'17*, volume 10625 of *LNCS*, pages 443–470. Springer, 2017. Full version: https://ia.cr/2017/798, Code: https://encrypto.de/code/UC.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC'87*, pages 218–229. ACM, 1987.

[GP81] Z. Galil and W. J. Paul. An efficient general purpose parallel computer. In *STOC'81*, pages 247–262. ACM, 1981.

[GVW13] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. In *STOC'13*, pages 545–554. ACM, 2013.

[HKK+14] Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J. Malozemoff. Amortizing garbled circuits. In *CRYPTO'14*, volume 8617 of *LNCS*, pages 458–475. Springer, 2014.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[IP07] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *TCC'07*, volume 4392 of *LNCS*, pages 575–594. Springer, 2007.

[Kő31] D. Kőnig. Gráfok és mátrixok. In *Matematikai és Fizikai Lapok*, volume 38, pages 116–119, 1931.

[KKW17] W. S. Kennedy, V. Kolesnikov, and G. T. Wilfong. Overlaying conditional circuit clauses for secure computation. In *ASIACRYPT'17*, volume 10625 of *LNCS*, pages 499–528. Springer, 2017.

[KM11] J. Katz and L. Malka. Constant-round private function evaluation with linear complexity. In *ASIACRYPT'11*, volume 7073 of *LNCS*, pages 556–571. Springer, 2011.

[Kol18] V. Kolesnikov. Free IF: How to omit inactive branches and implement S-universal garbled circuit (almost) for free. In *ASIACRYPT'18*, volume 11274 of *LNCS*, pages 34–58. Springer, 2018.

[KR11] S. Kamara and M. Raykova. Secure outsourced computation in a multi-tenant cloud. In *IBM Workshop on Cryptography and Security in Clouds, Zürich, Switzerland*, 2011.

[KS08a] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP'08*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[KS08b] V. Kolesnikov and T. Schneider. A practical universal circuit construction and secure evaluation of private functions. In *FC'08*, volume 5143 of *LNCS*, pages 83–97. Springer, 2008. Code: https://encrypto.de/code/FairplayPF.

[KS16] Á. Kiss and T. Schneider. Valiant's universal circuit is practical. In *EUROCRYPT'16*, volume 9665 of *LNCS*, pages 699–728. Springer, 2016. Full version: https://ia.cr/2016/093, Code: https://encrypto.de/code/UC.

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

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

[LP09b] L. Lovász and M. D. Plummer. *Matching Theory*. AMS Chelsea Publishing Series. American Mathematical Soc., 2009.

[LR15] Y. Lindell and B. Riva. Blazing fast 2PC in the offline/online setting with security for malicious adversaries. In *CCS'15*, pages 579–590. ACM, 2015.

[Mey83] F. Meyer auf der Heide. Efficiency of universal parallel computers. In *Theoretical Computer Science*, volume 145 of *LNCS*, pages 221–241. Springer, 1983.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – A secure two-party computation system. In *USENIX Security'04*, pages 287–302. USENIX, 2004.

[MR17] P. Mohassel and M. Rosulek. Non-interactive secure 2PC in the offline/online and batch settings. In *EUROCRYPT'17*, volume 10212 of *LNCS*, pages 425–455. Springer, 2017.

[MS13] P. Mohassel and S. S. Sadeghian. How to hide circuits in MPC – An efficient framework for private function evaluation. In *EUROCRYPT'13*, volume 7881 of *LNCS*, pages 557–574. Springer, 2013.

[MSS14] P. Mohassel, S. S. Sadeghian, and N. P. Smart. Actively secure private function evaluation. In *ASIACRYPT'14*, volume 8874 of *LNCS*, pages 486–505. Springer, 2014.

[NPS99] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Electronic Commerce (EC'99)*, pages 129–139, 1999.

[NSMS14] S. Niksefat, B. Sadeghiyan, P. Mohassel, and S. S. Sadeghian. ZIDS: A privacy-preserving intrusion detection system using secure two-party computation protocols. *Comput. J.*, 57(4):494–509, 2014.

[OI05] R. Ostrovsky and W. E. Skeith III. Private searching on streaming data. In *CRYPTO'05*, volume 3621 of *LNCS*, pages 223–240. Springer, 2005.

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

[PKV+14] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. Geol Choi, W. George, A. D. Keromytis, and S. Bellovin. Blind Seer: A scalable private DBMS. In *IEEE S&P'14*, pages 359–374. IEEE, 2014.

[PSS09] A. Paus, A.-R. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In *ACNS'09*, volume 5536 of *LNCS*, pages 89–106. Springer, 2009.

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

[Sha49] C. Shannon. The synthesis of two-terminal switching circuits. *Bell Labs Technical Journal*, 28(1):59–98, 1949.

[SYY99] T. Sander, A. L. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *FOCS'99*, pages 554–567. IEEE, 1999.

[SZ13] Thomas Schneider and Michael Zohner. GMW vs. Yao? efficient secure two-party computation with low depth circuits. In *Financial Cryptography and Data Security (FC'13)*, volume 7859 of *LNCS*, pages 275–292. Springer, 2013.

[TS18] S. Tillich and N. Smart. Circuits of basic functions suitable for MPC and FHE, 2018. https://homes.esat.kuleuven.be/~nsmart/MPC/.

[Val76] L. G. Valiant. Universal circuits (preliminary report). In *STOC'76*, pages 196–203. ACM, 1976.

[Wak68] A. Waksman. A permutation network. *J. ACM*, 15(1):159–163, 1968.

[Weg87] I. Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987.

[Yao82] A. C.-C. Yao. Protocols for secure computations (extended abstract). In *FOCS'82*, pages 160–164. IEEE, 1982.

[Yao86] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS'86*, pages 162–167. IEEE, 1986.

[Zim15] J. Zimmerman. How to obfuscate programs directly. In *EUROCRYPT'15*, volume 9057 of *LNCS*, pages 439–467. Springer, 2015.

[ZYZL18] Shuoyao Zhao, Yu Yu, Jiang Zhang, and Hanlin Liu. Valiant's universal circuits revisited: an overall improvement and a lower bound. Cryptology ePrint Archive, Report 2018/943, 2018. https://ia.cr/2018/943.

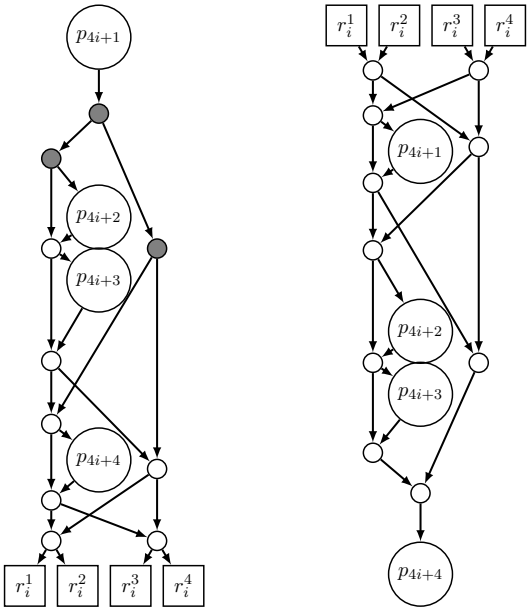
A Abbreviations and Notations

ABE	Attribute-based encryption.
DAG	Directed acyclic graph.
DBMS	Database management system.
EUG	Edge-universal graph.
GRR3	Garbled row reduction.
OT	Oblivious transfer.
PFE	Private function evaluation.
semi-PFE	Semi-private function evaluation.
SFE	Secure function evaluation or secure two-party computation.
UC	Universal circuit.

f	Function to be privately evaluated using a universal circuit.
c^f	Control bits for a universal circuit to compute function f.
u	Number of inputs in simulated Boolean circuit.
v	Number of outputs in simulated Boolean circuit.
$\hat{g}$	Number of gates in simulated Boolean circuit with arbitrary fanin and fanout.
$\tilde{g}$	Number of gates in simulated Boolean circuit with fanin 2 and arbitrary fanout.
g	Number of gates in simulated Boolean circuit with fanin and fanout 2. $\tilde{g} \leq g \leq \tilde{g} + v$.
$C_{u,v}^{\hat{g}}$	The Boolean circuit that describes f with arbitrary fanin and fanout.
$C_{u,v}^{\tilde{g}}$	The Boolean circuit that describes f with fanin 2 and arbitrary fanout.
$C_{u,v}^{g}$	The Boolean circuit that describes f with fanin and fanout 2.
n	Size of the simulated circuit $C_{u,v}^{g}$ with fanin and fanout 2, $n = u + v + g$.
d	Depth of the simulated circuit $C_{u,v}^{g}$.
G	The $\Gamma_2(n)$ graph of $C_{u,v}^{g}$ where every input, output and gate is represented with a node and every wire is represented with an edge.
$\Gamma_\rho(n)$	The set of all graphs with fanin and fanout ρ and n nodes.
$U_n(\Gamma_\rho)$	Edge-universal graph for $\Gamma_\rho(n)$ graphs, used generally for Valiant's construction.
$U_n^{(k)}(\Gamma_\rho)$	k-way edge-universal graph for $\Gamma_\rho(n)$ graphs.
$U_n^{\text{hybrid}(K)}(\Gamma_\rho)$	Hybrid edge-universal graph for $\Gamma_\rho(n)$ graphs with a set K of k possible values, e.g., $K = \{2, 4\}$.
p_i	Distinguished nodes in $U_n(\Gamma_\rho)$, called poles, with fanin and fanout ρ.
P	Set of all poles in $U_n(\Gamma_\rho)$.
U	A universal gate that computes any function with two inputs and one output, using four control bits c_0, c_1, c_2, c_3 as in Eq. 1.
X	A two-output X-switching block that returns its two input values either in the same or in reversed order depending on control bit c.
Y	A one-output Y-switching block that returns one of the two input values depending on control bit c.
$B^{(k)}$	Body block of k-way EUG.
$P^{(k)}$	Permutation network for k nodes.
$P_{\text{l}}^{(k)}$	Lower bound on the size of the permutation network for k nodes.
$P_{\text{W}}^{(k)}$	Size of the Waksman's permutation network [Wak68] for k nodes.
U_n^{KS08}	The UC of [KS08b].
$\text{UC}^{\text{Valiant-2}}$	Valiant's 2-way UC [Val76].
$\text{UC}^{\text{Valiant-4}}$	Valiant's 4-way UC [Val76].
$\text{UC}^{\text{Zhao et al.-4}}$	Valiant's 4-way UC with Zhao et al.'s optimization [ZYZL18].
$\text{UC}^{\text{H(Valiant-2,4)}}$	Hybrid UC with $\text{UC}^{\text{Valiant-2}}$ and $\text{UC}^{\text{Valiant-4}}$.
$\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$	Hybrid UC with $\text{UC}^{\text{Valiant-2}}$ and $\text{UC}^{\text{Zhao et al.-4}}$.

B Optimized Blocks for Zhao et al.'s 4-Way UC

In this section, we depict the head and tail block constructions in Fig. 18a and Fig. 18b, respectively, for Zhao et al.'s body block (cf. Fig. 4b), similar to those of [GKS17, Figs. 4e-4f] for Valiant's 4-way UC. Similarly, tail blocks can be designed also for smaller number of poles in the final block, but as shown in Table 6, they will have the same size as our tail blocks for Valiant's 4-way UC [GKS17, Figs. 4g-4i].



(a) Head block. (b) Tail block (4).

Figure 18: Optimized blocks for Zhao et al.'s 4-way block (Fig. 4b) [ZYZL18].

C Example Output of Our UC Compiler

In this section, we provide an example output of our UC compiler, i.e., the circuit and programming files shown on Listings 3 and 4 corresponding to the universal circuit shown in Fig. 1e on p. 10.

Listing 3: Example output UC.

```
1  C 0 1
2  X 0 1 2
3  X 1 0 3
4  X 0 2 4
5  X 3 0 5
6  U 2 3 6
7  X 4 6 7
8  X 6 5 8
9  Y 4 7 9
10 Y 8 5 10
11 U 7 8 11
12 Y 9 11 12
13 Y 11 10 13
14 Y 12 13 14
15 O 14
```

Listing 4: Example programming.

```
1        //input bits
2  0     //X switch (no swap grey)
3  1     //X switch (swap green)
4  1     //X switch (swap blue)
5  0     //X switch (undefined)
6  1     //AND gate (0001)
7  1     //X switch (swap blue)
8  0     //X switch (no swap red)
9  0     //Y switch (undefined)
10 0     //Y switch (undefined)
11 6     //XOR gate (0110)
12 0     //Y switch (right input orange)
13 0     //Y switch (undefined)
14 1     //Y switch (left input orange)
15       //output bits
```

D Concrete Performance Measures for Private Function Evaluation

In this section, we provide the concrete performance measures used for depicting the runtimes and communication of PFE by securely evaluating UCs generated with $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ in Figs. 14 and 15, respectively.

Input circuit size n	10	100	1 000	10 000	38 518 (AES)	100 000	201 206 (SHA-256)	1 000 000
Yao LAN (s)	0.006	0.013	0.08	0.48	2.25	6.63	13.70	78.73
GMW LAN (s)	0.032	0.386	3.89	38.92	147.14	389.85	783.94	3 925.94
Yao WAN (s)	0.323	0.413	0.75	2.81	10.71	29.49	63.10	354.32
GMW WAN (s)	2.000	23.990	240.55	2395.32	9044.30	*	*	*
Yao comm. (MB)	0.005	0.087	1.51	21.79	99.36	287.51	620.07	3 562.21
GMW comm. (MB)	0.013	0.182	2.27	27.20	114.21	319.31	670.23	3 660.17
GMW rounds	34	441	4 491	44 991	169 871	449 991	903 686	4 499 991

Table 7: Runtime and communication of PFE with universal circuits generated for input circuit size n (cf. Table 2 for the respective UC sizes). * denotes cases where an experiment would have taken more than 5 hours and therefore was not performed.