More Efficient Universal Circuit Constructions
(Full Version)*

Daniel Günther, Ágnes Kiss, Thomas Schneider

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

Abstract. A universal circuit (UC) can be programmed to simulate any circuit up to a given size n by specifying its program bits. UCs have several applications, including private function evaluation (PFE). The asymptotical lower bound for the size of a UC is proven to be $\Omega(n \log n)$. In fact, Valiant (STOC'76) provided two theoretical UC constructions using so-called 2-way and 4-way constructions, with sizes $5n \log_2 n$ and $4.75n \log_2 n$, respectively. The 2-way UC has recently been brought into practice in concurrent and independent results by Kiss and Schneider (EUROCRYPT'16) and Lipmaa et al. (Eprint 2016/017). Moreover, the latter work generalized Valiant's construction to any k-way UC.

In this paper, we revisit Valiant's UC constructions and the recent results, and provide a modular and generic embedding algorithm for any k-way UC. Furthermore, we discuss the possibility for a more efficient UC based on a 3-way recursive strategy. We show with a counterexample that even though it is a promising approach, the 3-way UC does not yield an asymptotically better result than the 4-way UC. We propose a hybrid approach that combines the 2-way with the 4-way UC in order to minimize the size of the resulting UC. We elaborate on the concrete size and depth of all discussed UC constructions and show that our hybrid UC yields on average 3.65% improvement in size over the 2-way UC. We implement the 4-way UC in a modular manner based on our proposed embedding algorithm, and show that our methods for programming the UC can be generalized for any k-way construction.

Keywords: Universal circuit, private function evaluation, function hiding

1 Introduction

Universal circuits (UCs) are Boolean circuits that can be programmed to simulate any Boolean function $f(x)$ up to a given size by specifying a set of program bits p_f. The UC then receives these program bits as input besides the input x to the functionality, and computes the result as $UC(x, p_f) = f(x)$. This means that the same UC can evaluate multiple Boolean circuits, only the different program bits are to be specified.

Valiant proposed an asymptotically size-optimal construction in [Val76] with size $\Theta(n \log n)$ and depth $\mathcal{O}(n)$, where n is the size of the simulated Boolean circuit description of $f(x)$. He provides two constructions, based on 2-way and 4-way recursive structures. A depth-optimal construction with depth $\mathcal{O}(d)$ that simulates circuits with depth d was proposed in [CH85], but it has a significantly larger size of $\mathcal{O}(\frac{n^3 d}{\log n})$. Recently, optimizations of Valiant's size-optimized construction appeared in concurrent and independent works of [KS16] and [LMS16, Sad15]. Both works implement Valiant's 2-way recursive construction.

1.1 Applications of Universal Circuits

Size-optimized universal circuits have many applications. We review some of them here and refer to [KS16, LMS16] for further details.

*Please cite the conference version of this work published at Asiacrypt'17 [GKS17].

Private Function Evaluation (PFE). One of the most prominent applications of UCs is private function evaluation (PFE), firstly proposed in [AF90]. Secure two-party computation or secure function evaluation (SFE) provides interactive protocols for evaluating a public function $f(x, y)$ on two parties' private inputs x and y. However, in some scenarios, the function f is a secret input of one of the parties. This setting is called private function evaluation (PFE). PFE of $f(x)$ can be achieved by running SFE of $UC(x, p_f)$, where the UC is a public function and the program bits p_f – and therefore f – are kept private due to the properties of SFE. Protocols designed especially for PFE such as [MS13, BBKL17] achieve the same asymptotic complexity $\mathcal{O}(n \log n)$ as PFE using UCs, where n is the size of the function f.[1] However, to the best of our knowledge, they have not yet been implemented, and they are not as generally applicable as PFE with UCs.

UC-based PFE can be easily integrated into any SFE framework and can directly benefit from recent optimizations. For instance, the fact that SFE can be outsourced to two or more computing servers [KR11] implies that *outsourcing UC-based PFE* is also directly possible [KS16]. The non-interactive secure computation protocol of [AMPR14] can also be generalized to obtain a *non-interactive PFE* protocol [LMS16].

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. PFE allows for running *proprietary software* on private data, such as privacy-preserving software diagnosis [BPSW07], medical programs [BFK+09], or privacy-preserving intrusion detection [NSMS14]. UCs can be applied to obliviously *filter remote streaming data* [OI05] and for hiding queries in *private database management systems* such as Blind Seer [PKV+14, FVK+15].

Applications Beyond PFE. 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], which has been made round-optimal in [MR17]. It can be applied for efficient *verifiable computation* as described in [FGP14], and for multi-hop homomorphic encryptionwith function privacy [GHV10]. Ciphertext-policy *Attribute-Based Encryption* was proposed in [Att14], where the policy circuit is hidden [GGH+13b, GVW13].

1.2 Related Work on Universal Circuits

Valiant defined universal circuits in [Val76] and gave two size-optimized constructions. The constructions are based on so-called edge-universal graphs (EUGs) and utilize either a 2-way or a 4-way recursive structure, also called 2-way or 4-way UCs. Both achieve the asymptotically optimal size $\Theta(n \log n)$ [Val76, Weg87], where n is the size of the simulated circuit. Wegener was the first

[1] There also exist PFE protocols with linear complexity $\mathcal{O}(n)$ which are based on public-key primitives [KM11, MS13, MSS14]. However, the concrete complexity of these protocols is worse than that of the protocols based on (mostly) symmetric-key primitives, i.e., the OT-based PFE protocols of [MS13, BBKL17] or PFE using UCs.

to describe the key ideas behind Valiant's 2-way construction more clearly in [Weg87]. The concrete complexity of the 4-way UC is $\sim 4.75n \log_2 n$ which is smaller than that of the 2-way UC of $\sim 5n \log_2 n$ [Val76].

The first modular UC construction was proposed by Kolesnikov and Schneider in [KS08b]. This construction achieves a non-optimal asymptotic complexity of $\mathcal{O}(n \log^2 n)$. However, this was the first implementation of UCs and provides some efficient building blocks. A generalization of UCs for n-input gates was given by Sadeghi and Schneider in [SS08].

Recently, two independent works have optimized and implemented Valiant's 2-way UC [KS16, LMS16]. Kiss and Schneider in [KS16] mainly focus on the most prominent application of UCs, i.e., private function evaluation (PFE). Due to the free-XOR optimization of [KS08a] in the SFE setting, they optimize the size of the UC for the number of AND gates in the resulting UC implementation and provide a framework for PFE using UCs as public function. They also propose hybrid constructions for circuits with a large number of inputs and outputs, utilizing efficient building blocks from [KS08b]. Lipmaa et al. in [LMS16] also provide an (unpublished) implementation of the 2-way UC. While keeping the number of AND gates minimal, they additionally optimize for the total number of gates, i.e., include optimizations to also reduce the number of XOR gates. They adapt the construction to arithmetic circuits and generalize the design to a k-way construction in a modular manner, for $k \geq 2$.

Both papers utilize 2-coloring of the underlying graphs for defining the program bits p_f for any given functionality f. Generally, 2-coloring can be utilized for any 2^i-way construction. [LMS16] calculate the optimal value for k to be 3.147, and conclude that the two candidates for the most efficient 2^i-way constructions are the 2-way and 4-way UCs, of which the 4-way construction results in an asymptotically smaller size.

In conclusion, so far only Valiant's 2-way UC has been implemented and the not yet implemented 4-way construction was postulated to be the most efficient one. Moreover, the concrete sizes of k-way constructions for $k \neq 2$ and $k \neq 4$ have also not been investigated.

1.3 Outline and Our Contributions

In summary, we provide the first implementation and detailed evaluation of Valiant's 4-way UC and propose an even more efficient hybrid UC. We elaborate on the size of the generalized k-way UCs for $k \neq 2$ and $k \neq 4$.

After revisiting Valiant's UC construction [Val76, KS16] and its k-way generalization [LMS16] in §2, we provide the following contributions:

Our modular programming algorithm (§3): We detail a modular algorithm that provides the description of the input function f as program bits p_f to the UC, both for Valiant's 4-way UC as well as for the k-way UC of Lipmaa et al. [LMS16].

New universal circuit constructions (§4): We start with a new 3-way UC. After providing modular building blocks for this UC, we show that it is asymptotically larger than Valiant's UCs. Then, we propose a *hybrid UC construction* that can efficiently combine k-way constructions for multiple values of k.[2] With this, we combine Valiant's 2-way and 4-way UCs to achieve the smallest UC known so far.

[2] Our hybrid UC is orthogonal to that of [KS16], who combine Valiant's UC with building blocks from [KS08b] for the inputs and outputs.

n	Special-purpose PFE		UC-based PFE using Yao		
	[MS13]	[BBKL17]	2-way UC [KS16]	Our 4-way UC	Our Hybrid UC
10^3	3.5 MB	2.0 MB	0.6 MB	0.6 MB	0.6 MB
10^4	44.8 MB	26.3 MB	8.4 MB	8.4 MB	8.2 MB
10^5	549.6 MB	324.0 MB	109.6 MB	107.8 MB	106.2 MB
10^6	6 509.9 MB	3 847.9 MB	1 360.3 MB	1 308.4 MB	1 308.4 MB
10^7	75 236.5 MB	44 562.1 MB	16 038.8 MB	15 677.7 MB	15 413.7 MB

Table 1: Comparison of overall communication between special-purpose PFE protocols and UC-based ones for simulated circuits of size n. The numbers are for 128 bit symmetric security. The underlying SFE protocol for UC-based PFE is Yao's protocol [Yao86] with the garbled row reduction optimization [NPS99] and X- and Y-switching blocks are instantiated using free XORs as described in [KS08a]. This yields one ciphertext per X- and Y-switching block, and three ciphertexts per universal gate.

Size and depth of UCs (§5): We compare the asymptotic and concrete sizes and depths of Valiant's (2-way and 4-way) UC constructions and that of different k-way UCs. We show that of all k-way UCs, Valiant's 4-way UC provides the best results for large circuits. Moreover, our hybrid UC in most cases improves over the 2-way UC by up to around 4.5% in its size, and over the 4-way UC by up to 2% (for large input circuits). In Table 1 we compare the concrete communication of PFE using SFE and our new UC implementation to the previous works on special-purpose OT-based PFE protocols.

Implementation of Valiant's 4-way UC and experiments (§6): We implement Valiant's 4-way UC and describe how our implementation can directly be used in the PFE framework of [KS16]. We experimentally evaluate the performance of our UC generation and programming algorithm with a set of example circuits and compare it on the same platform with the 2-way UC compiler of [KS16]. We also discuss the modifications needed to be included for implementing our hybrid construction.

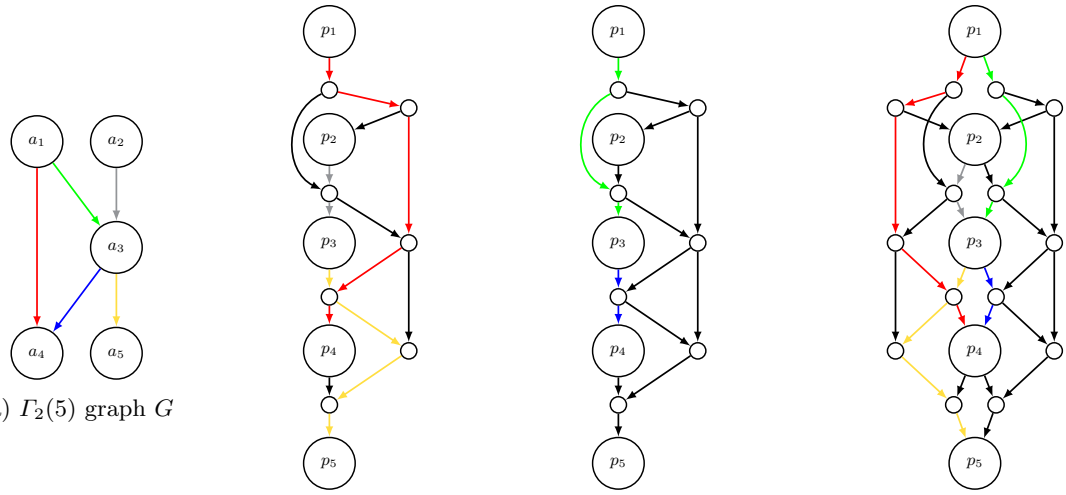
2 Preliminaries

In this section, we summarize the existing UC constructions. We provide necessary background information in §2.1, explain Valiant's construction [Val76] in §2.2 and the improvements of [KS16, LMS16] on the 2-way, 4-way and k-way UCs in §2.3, §2.4 and §2.5, respectively.

2.1 Preliminaries to Valiant's UC Constructions

Let $G = (V, E)$ be a *directed graph* with set of *nodes* V and *edges* $E \subseteq V \times V$. The number of incoming [outgoing] edges of a node is called its *indegree* [*outdegree*]. A graph has *fanin* [*fanout*] d if the indegree [outdegree] of all its nodes is at most d. In the following, we denote by $\Gamma_d(n)$ the set of all acyclic graphs with fanin and fanout d having n nodes. Similarly, the fanin [fanout] of a circuit can be defined based on the maximal number of incoming [outgoing] wires of all its gates, inputs and outputs.

Let $G = (V, E) \in \Gamma_d(n)$. A mapping $\eta^G : V \to \{1, \ldots, n\}$ is called *topological order* if $(a_i, a_j) \in E \Rightarrow \eta^G(a_i) < \eta^G(a_j)$ and $\forall a_1, a_2 \in V : \eta^G(a_1) = \eta^G(a_2) \Rightarrow a_1 = a_2$. A topological order in $G \in \Gamma_d(n)$ can be found with computational complexity $\mathcal{O}(dn)$.



(a) $\Gamma_2(5)$ graph G (b) $U_5(\Gamma_1)_1$ (c) $U_5(\Gamma_1)_2$ (d) EUG $U_5(\Gamma_2)$

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

A circuit $C^{k^*}_{u,v}$ with u inputs, k^* gates and v outputs and fanin or fanout $d > 2$ can be reduced to a circuit with fanin and fanout 2. Shannon's expansion theorem [Sha49, Sch08] describes how gates with larger fanin can be reduced to gates with two inputs by adding additional gates. [Val76, KS16] describe the process of adding copy gates in order to eliminate larger fanout and elaborate on the implied overhead ($k \leq 2k^* + v$). [KS08b, KS16] implement these methods and we thus assume that our input Boolean circuit $C^k_{u,v}$ has fanin and fanout 2 for all its u inputs, k gates and v outputs. We transform $C^k_{u,v}$ with u inputs, v outputs and k gates into a $\Gamma_2(n)$ graph G with $n = u + v + k$ by creating a node for each input, gate and output, and an edge for each wire in $C^k_{u,v}$.

Edge-embedding is a mapping from graph $G = (V, E)$ into $G' = (V', E')$ with $V \subseteq V'$ and E' containing a path for each $e \in E$, such that the paths are pairwise edge-disjoint. A graph $U_n(\Gamma_d) = (V_U, E_U)$ is an *Edge-Universal Graph* (EUG) for $\Gamma_d(n)$ if every graph $G \in \Gamma_d(n)$ can be edge-embedded into $U_n(\Gamma_d)$.[3] $U_n(\Gamma_d)$ has distinguished nodes called *poles* $\{p_1, \ldots, p_n\} \subseteq V_U$ where each node $a \in V$ is mapped to exactly one pole with a mapping φ, such that every node in G has a corresponding pole in $U_n(\Gamma_d)$. This mapping is defined by a concrete topological order η^G of the original graph G, i.e., $\varphi : V \to V_U$ with $\varphi(a) = p_{\eta^G(a)}$. Besides the poles, $U_n(\Gamma_d)$ might have additional nodes that enable the edge-embedding. For each edge $(a_i, a_j) \in E$ we then define a disjoint path between the corresponding poles $(\varphi(a_i), \ldots, \varphi(a_j)) = (p_{\eta^G(a_i)}, \ldots, p_{\eta^G(a_j)})$ in $U_n(\Gamma_d)$, i.e., without using any edge in $U_n(\Gamma_d)$ in more than one path.

Let $U_n(\Gamma_1)$ be an EUG for graphs in $\Gamma_1(n)$ with poles $P = \{p_1, \ldots, p_n\}$. We require that the poles have fanin and fanout 1, while all other nodes have fanin and fanout 2. An EUG $U_n(\Gamma_d)$ for $d \geq 2$ can be created by taking d instances of $U_n(\Gamma_1)$ EUGs, and merging each pole p_i with its multiple instances, allowing the poles to have fanin-fanout d. Let $U_n(\Gamma_d) = (V'_U, E'_U)$ be an EUG with fanin and fanout d, constructed with $U_n(\Gamma_1)_1 = (V_1, E_1), \ldots, U_n(\Gamma_1)_d = (V_d, E_d)$. P contains

[3] For the sake of simplicity, we denote this graph with $U_n(\Gamma_d)$ instead of $U(\Gamma_d(n))$.

(a) X-Switching Block (b) Y-Switching Block

Fig. 2: Switching Blocks

the merged poles and $V'_U = P \cup_{i=1}^{d} V_i \backslash P_i$ and $E'_U = \cup_{i=1}^{d} E_i$. Now, every pole in $U_n(\Gamma_d)$ has at most d, 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 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. 1b and $U_5(\Gamma_1)_2$ in Fig. 1c. The edges (a_1, a_4), (a_2, a_3) and (a_3, 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. 1d.

2.2 Valiant's UC Constructions

The size of a function f represented by a circuit $C^k_{u,v}$ with fanin and fanout 2 is $n = u + v + k$. In the following, we describe Valiant's UC construction [Val76, Weg87] that can be programmed to evaluate any function of size n. Circuit $C^k_{u,v}$ is represented as a graph $G \in \Gamma_2(n)$ (cf. §2.1).

Valiant's UC is based on an EUG $U_n(\Gamma_2) = (V_U, E_U)$ with fanin and fanout 2, which can be transformed to a Boolean circuit. $P \subseteq V_U$ contains the poles of $U_n(\Gamma_2)$ (cf. §2.1). These poles correspond to the inputs, gates and outputs of $C^k_{u,v}$, i.e., poles $\{1, \ldots, u\}$ correspond to the inputs, $\{(u+1), \ldots, (u+k)\}$ to the gates, $\{(u+k+1), \ldots, n\}$ to the outputs. The edges of the graph of the circuit $G = (V, E)$ have to be embedded into $U_n(\Gamma_2)$. After the transformations described in §2.1, every node in G has fanin and fanout 2, and we denote a topological order on V by η^G. We briefly describe the edge-embedding process in §2.3 and §3.

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

G1 If w is a pole and corresponds to an input or output in G, then w is an *input or output* in $U_n(\Gamma_2)$ as well.

G2 If w is a pole and corresponds to a gate in G, w is programmed as a *universal gate* (UG). A 2-input UG supports any of the 16 possible gate types represented by the 4 control bits of the gate table (c_1, c_2, c_3, c_4). It implements function ug: $\{0,1\}^2 \times \{0,1\}^4 \to \{0,1\}$ that computes:

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

Generally, a UG can be implemented with 3 AND and 6 XOR gates (resp. with a two-input gate when using Yao's protocol for SFE) [KS16].

G3 If w is no pole and has indegree and outdegree 2, w is programmed as an *X-switching block*, that computes $f_X : \{0,1\}^2 \times \{0,1\} \to \{0,1\}^2$ with $f_X((x_1, x_2), c) = (x_{1+c}, x_{2-c})$ 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. This block can be implemented with 1 AND and 3 XORs (resp. a one-input gate with Yao) [KS08a].

G4 If w is no pole and has indegree 2 and outdegree 1, w is programmed as a *Y-switching block* that computes $f_Y : \{0,1\}^2 \times \{0,1\} \to \{0,1\}$ with $f_Y((x_1, x_2), c) = x_{1+c}$ as visualized in Fig. 2b. The inputs of a Y-switching block are forwarded to its output depending on the control bit c. A Y-switching block provides the functionality of a 2-input multiplexer, and can be implemented with 1 AND and 2 XORs (resp. a one-input gate with Yao) [KS08a].

G5 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 itself does not have 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.

G6 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.

The nodes programmed as UG (**G2**), X-switching block (**G3**) or Y-switching block (**G4**) are so-called programmable blocks. This means that a programming bit or vector is necessary besides the two inputs to define their behavior as described above. These programming bits and vectors that build up the programmig of the UC p_f are defined by the paths in the edge-embedding of G (the graph of circuit $C^k_{u,v}$ describing f) into $U_n(\Gamma_2)$.

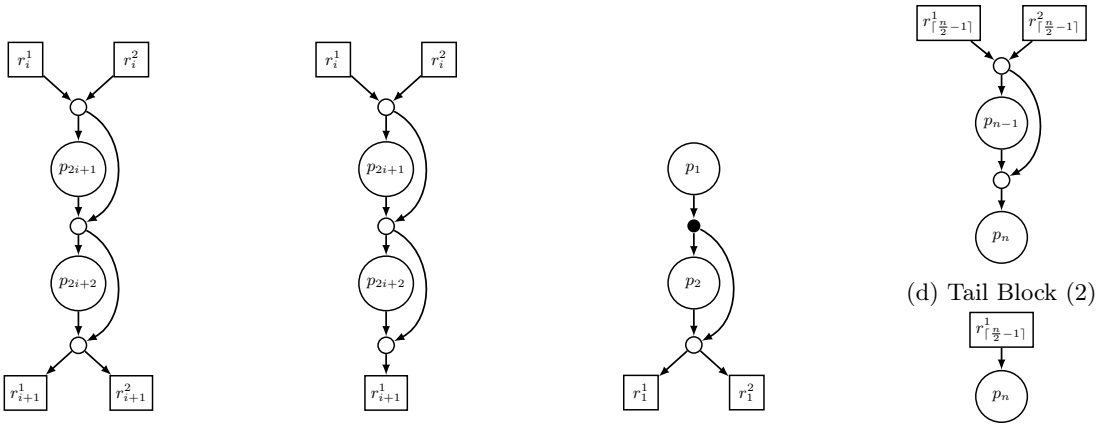
Recursion Base. Valiant's construction is recursive, and the recursion base is reached when the number of poles is between 1 and 6. These recursion base graphs are shown in [Val76, KS16]. $U_1(\Gamma_1)$ is a single pole, $U_2(\Gamma_1)$ and $U_3(\Gamma_1)$ are two and three connected poles, respectively. $U_4(\Gamma_1)$ is constructed with 3 additional nodes which are between 2 poles, i.e. there is alternatively a pole and a node from top to bottom. Valiant provides hand-optimized EUG constructions for $U_5(\Gamma_1)$ and $U_6(\Gamma_1)$ [Val76], with 7 and 9 additional nodes, respectively.

2.3 Valiant's 2-Way UC Construction

We described in §2.1 that a $U_n(\Gamma_d)$ EUG can be constructed of d instances of $U_n(\Gamma_1)$ EUGs. Therefore, Valiant provides an EUG for $\Gamma_1(n)$ graphs, two of which can build an EUG for $\Gamma_2(n)$ graphs. Let $P = \{p_1, \ldots, p_n\}$ be the set of poles in $U_n^{(2)}(\Gamma_1)$ that have indegree and outdegree 1. Valiant's 2-way EUG construction for $\Gamma_1(n)$ graphs of size $\sim 2.5n \log_2 n$ is shown in Fig. 3, where we emphasize the poles as large circles and the additional nodes as small circles or rectangles. The corresponding UC has twice the size $\sim 5n \log_2 n$, since it corresponds to the EUG for $\Gamma_2(n)$ graphs.

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 $R^2_{\lceil \frac{n}{2}-1 \rceil} = \{r^2_1, \ldots r^2_{\lceil \frac{n}{2}-1 \rceil}\}$. Another EUG is 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 this level.

This construction is called the *2-way EUG or UC construction* since there are two sets of recursion nodes at each recursion step. An open-source implementation of this construction optimized for PFE is provided in [KS16]. Independently, [LMS16] also implemented this 2-way UC, additionally optimizing for the total number of gates.



(a) Body Block (2) (b) Body Block (1) (c) Head Block (d) Tail Block (2) (e) Tail Block (1)

Fig. 3: Fig. 3a shows Valiant's 2-way EUG $U_n^{(2)}(\Gamma_1)$ [Val76]. Fig. 3c shows the corresponding head block, Fig. 3b and Figs. 3d–3e show body and tail blocks, respectively, for different numbers of poles.

2.4 Valiant's 4-Way UC Construction

Valiant provides another, so-called *4-way EUG or UC construction* [Val76]. $U_n^{(4)}(\Gamma_1)$ 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 $R^4_{\lceil \frac{n}{4}-1 \rceil}$ are the poles in the next recursion step (cf. Fig. 5a on p. 10). The recursion base is the same as for the 2-way UC construction described in §2.2. This construction results in UCs of smaller size $\sim 4.75n \log_2 n$ but has not been implemented before due to its more complicated structure and programming algorithm.

2.5 Lipmaa et al.'s Generalized k-Way UC Construction

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 construction*. We note that their construction slightly differs from Valiant's EUG construction, 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. §2.2). 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.

The idea is to split $n = u + v + k$ in $m = \lceil \frac{n}{k} \rceil$ blocks as shown in Fig. 4. 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 every 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 $U_n^{(k)}(\Gamma_2)$ to the problem of finding the smallest EUG $U_k(\Gamma_1)$. Their solution is to build the block-intern EUG with the UC construction of [KS08b], which was claimed to be

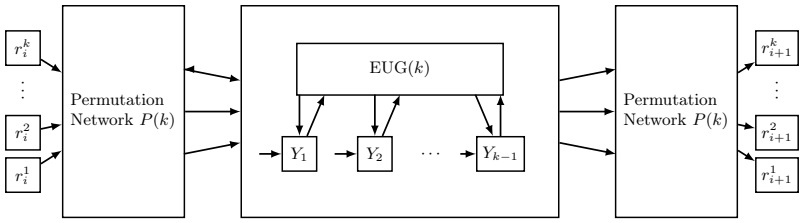


Fig. 4: k-way EUG construction $U_n^{(k)}(\Gamma_1)$ [LMS16].

more efficient for smaller circuits than [Val76]. However, they calculate the optimal k value to be around 3.147, 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].

3 Our Modular Edge-Embedding Algorithm

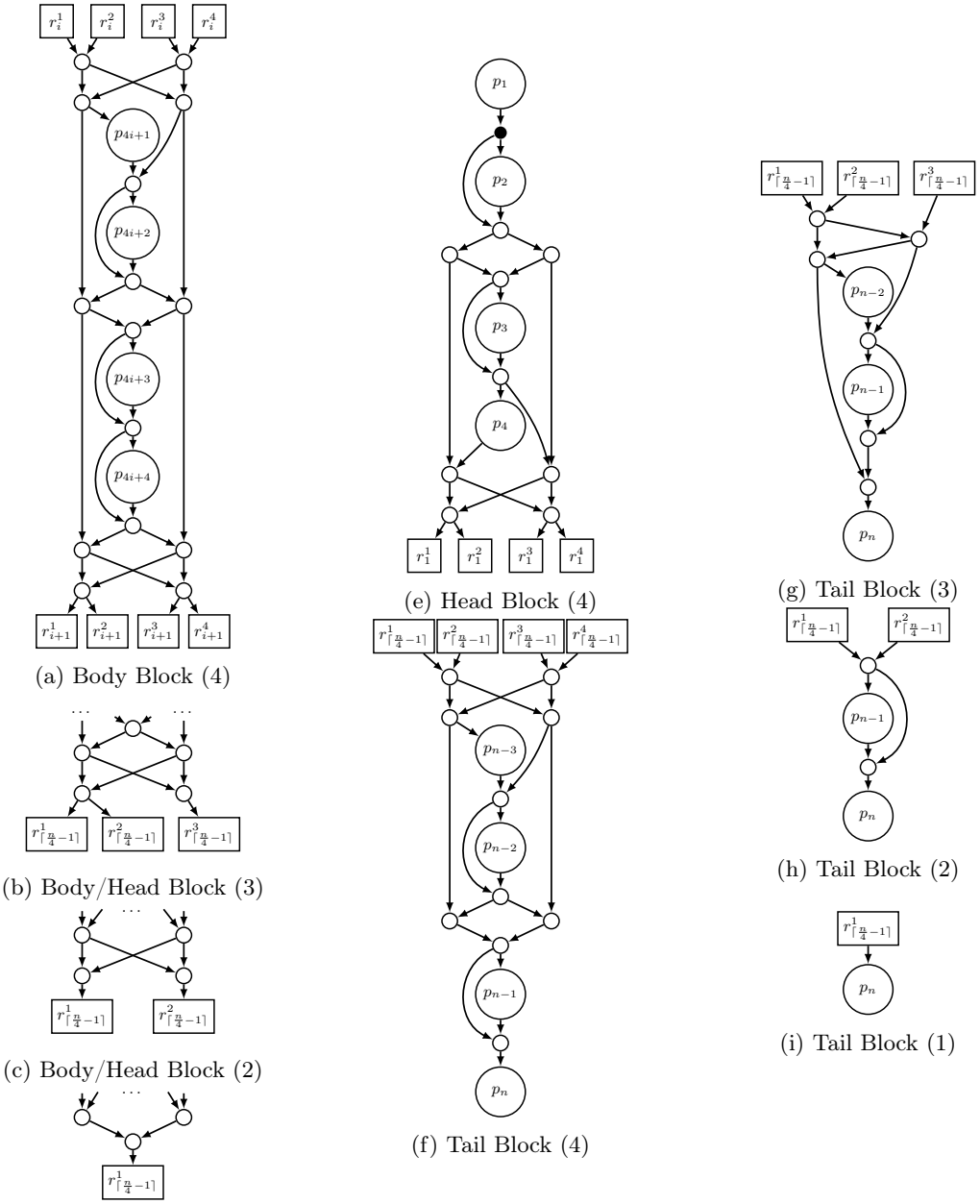
The detailed embedding algorithm and the open-source UC implementation of [KS16] was specifically built for the 2-way UC, dealing with the whole UC skeleton as one block. In contrast, based on the modular design of [LMS16], we modularize the edge-embedding task into multiple sub-tasks and describe how they can be performed separately. In this section, we detail this modular approach for edge-embedding a graph into Valiant's 4-way EUG: the edge-embedding can be split into two parts, which are then combined. In §3.1, we describe our modular approach based on the edge-embedding algorithm of [KS16] for Valiant's 2-way EUG construction. This can be generalized to any 2^i-way EUG construction. Moreover, the same modular edge-embedding algorithm can be applied with a few modifications for Lipmaa et al.'s k-way recursive generalization [LMS16], which we describe in §3.2.

3.1 Edge-Embedding in Valiant's 4-Way UC

Similar to the 2-way EUG construction (cf. §2.3), Valiant provides a 4-way EUG construction for $\Gamma_1(n)$ graphs which can be extended to an EUG for $\Gamma_2(n)$ graphs by utilizing two instances $U_n^{(4)}(\Gamma_1)_1$ and $U_n^{(4)}(\Gamma_1)_2$ as described in §2.1. The construction with our optimizations is visualized in Fig. 5. Valiant offers the main, so-called *Body Block* (Fig. 5a) consisting of 4 poles (large circles), 15 nodes (small circles) as well as 8 recursion points (squares). These 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., $R^1_{\lceil \frac{n}{4}-1 \rceil} = \{r_1^1, \ldots, r^1_{\lceil \frac{n}{4}-1 \rceil}\}$, $R^2_{\lceil \frac{n}{4}-1 \rceil} = \{r_1^2, \ldots, r^2_{\lceil \frac{n}{4}-1 \rceil}\}$, $R^3_{\lceil \frac{n}{4}-1 \rceil} = \{r_1^3, \ldots, r^3_{\lceil \frac{n}{4}-1 \rceil}\}$, and $R^4_{\lceil \frac{n}{4}-1 \rceil} = \{r_1^4, \ldots, r^4_{\lceil \frac{n}{4}-1 \rceil}\}$ 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. §2.2.

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 provide so-called Head and Tail Blocks. A *Head Block* occurs at the top of a chain of blocks (cf. Fig. 5e), it has 4 poles, no inputs, 4 output recursion points and 10 nodes, of which the first one (denoted by a filled circle) has one input and therefore translates to wires in the UC.

As a counterpart, *Tail Blocks* occur at the bottom of a chain of blocks, have at most 4 poles, 4 input recursion points, no outputs and at most 10 nodes depending on the number of poles. The



(a) Body Block (4)

(b) Body/Head Block (3)

(c) Body/Head Block (2)

(d) Body/Head Block (1)

(e) Head Block (4)

(f) Tail Block (4)

(g) Tail Block (3)

(h) Tail Block (2)

(i) Tail Block (1)

Fig. 5: Fig. 5a shows Valiant's 4-way EUG $U_n^{(4)}(I_1)$ [Val76]. Fig. 5e shows our head block construction, Figs. 5a–5d and Figs. 5f–5i show our body and tail block constructions, respectively, for different numbers of poles.

4 tail block constructions are depicted in Figs. 5f–5i and are used, based on the remainder of n modulo 4, with the respective body or head blocks when $n \in \{5, 6, 7\}$, the lower parts of which are shown in Figs. 5a–5d.

Block Edge-Embedding. In this first part of the edge-embedding process, we consider the 4 top [bottom] recursion points of the block as intermediate nodes where the inputs [outputs] of the block enter [leave]. The blocks are built s.t. any of these inputs can be forwarded to exactly one of the 4 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 $U_n^{(4)}(\Gamma_1) = (V_U, E_U)$, let B be the block visualized in Fig. 5a with poles $p_{4i+1}, \ldots, p_{4i+4}$. Let mapping $\eta^U : V_U \to \mathbb{N}^+$ denote a topological order of all nodes and poles in V_U. Then, the nodes $r_i^1, \ldots, r_i^4$ and $r_{i+1}^1, \ldots, r_{i+1}^4$ denote the input and output recursion points of block B. Additionally, let $in = (in_1, \ldots, in_4) \in \{0, \ldots, 4\}^4$ and $out = (out_1, \ldots, out_4) \in \{0, \ldots, 7\}^4$ denote the input and output vectors of B. The value 0 of the input and output vectors is a *dummy value* which is used if an input [a pole] is not forwarded to any pole [output] of B. 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, 2 and 3 for poles p_2, p_3 and p_4 and values 4, 5, 6 and 7 for the output recursion points. Pole p_1 cannot be a destination for a path in B, since $\eta^U(p_1)$ is less than the topological order of any other pole in B. 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. 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. Then, $\mathcal{P}_B \subseteq \mathcal{P}$ denote the paths that are to be edge-embedded (cf. 6.1).

$$\forall l \in \{1, \ldots, 4\} : in_l \neq 0 \rightarrow (r_i^l, p_{in_l}) \in \mathcal{P}_B, \tag{2}$$

$$out_l \neq 0 \wedge out_l < 4 \rightarrow (p_j, p_{1+out_l}) \in \mathcal{P}_B \wedge \eta^U(p_j) < \eta^U(p_{1+out_l}), \tag{3}$$

$$out_l > 3 \rightarrow (p_j, r_{i+1}^{l-3}) \in \mathcal{P}_B. \tag{4}$$

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

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

Recursion Point Edge-Embedding. The block edge-embedding covers only the programming of the nodes within a block. Another task left 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.

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 §2.1, 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 $(U_n^{(4)}(\Gamma_1))_1$ and $(U_n^{(4)}(\Gamma_1))_2$, 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, LP09].

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:

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

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

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

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 $G_m = (V', E') \in \Gamma_2(\lceil \frac{n}{2} \rceil)$ be a graph with nodes $v'_1, \ldots, v'_{\lceil \frac{n}{2} \rceil}$. 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 $\theta_V(v_i) = v'_{\lceil \frac{i}{2} \rceil}$. That means 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 $\theta_E((v_i, v_j)) = (v_{\eta_{\text{in}}(\theta_V(v_i))}, v_{\eta_{\text{out}}(\theta_V(v_j))})$. That means 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 $e' = (v'_k, v'_l) \in E'$, s.t. $v'_k = \theta_V(v_i)$, but v'_l is not the new node of v_j in G_m but v'_{l+1}. $G_m = (V', E')$ 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 4-way 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 $U_n^{(4)}(\Gamma_1)$ (for G_2 it is the same).

For embedding in a 2-way UC, G_1 is firstly merged to a $\Gamma_2(\lceil \frac{n}{2} \rceil)$ graph G_m. G_m is then 2-colored and split into two $\Gamma_1(\lceil \frac{n}{2} \rceil)$ 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 Γ_1 subgraphs have at most 4 nodes.

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 with 4 or less nodes.

Listing 1: Edge-embedding algorithm for Valiant's 4-way EUG

```
1  procedure edge-embedding (U, G_1 = (V,E))
2   Let S be the set of the 4 Γ_1 subgraphs of G_1 in the supergraph
3   Let R be the 4 recursion step graphs
4   Let B be the set of blocks in U
5   for all e = (v_i, v_j) ∈ E do
6     Let i' and j' denote the positions of v_i and v_j in their blocks
7     b_i ← ⌈i/4⌉, b_j ← ⌈j/4⌉ // number of block in which v_i and v_j are
8     Let out [r_1] denote the output vector [recursion points] of B_{b_i}
9     Let in [r_0] denote the the input vector [recursion points] of B_{b_j}
10    if b_i = b_j do // v_i and v_j are in the same block
11          if v_i ≠ v_j do
12              out_{i'} ← j' − 1
13          end if
14    else // v_i and v_j are in different blocks
15          Let s = (V', E') ∈ S denote the Γ_1 graph with e' = (p_{b_i}, p_{b_{j−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 b_j = b_i + 1 do // b_j and b_i 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 + 4, in_x ← j'
26    end if
27  end for
28  Edge-embed all blocks in U // edge-embed all sub-blocks
29  for i = 1 to 4 do
30    if S_i exists do
31     call edge-embedding(R_i, S_i)
32    end if
33  end for
34 end procedure
```

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

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

1. **v_i and v_j are in the same block: $b_i = b_j$.** The edge-embedding can be solved within the block and no recursion points have to be programmed for this path. Therefore, vector *out* of block B_{b_i} is set accordingly.
2. **v_i and v_j are in different blocks: $b_i \neq b_j$.** There is an edge $e' = (b_i, b_{j-1})$ in one of the four $\Gamma_1(\lceil \frac{n}{4} - 1 \rceil)$ 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 $s \in S$

the subgraph of G_1 which contains e', and x denotes its number in S, i.e. $S_x = s$. This implies in which of the 4 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 programming 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 B_{b_i} and 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 programming bit thus has to be 0. The output vector of block B_{b_i} is the i'^{th} value to the x^{th} recursion point and the input vector of block 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 in- and output vectors of all blocks in B are set, they can be embedded with the block edge-embedding. For all 4 subgraphs of G_1 in the supergraph and in the EUG, we call the same procedure with $S_i \in S$, $R_i \in R$, $1 \leq i \leq 4$.

3.2 Edge-Embedding in Lipmaa et al.'s k-way 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 described in §2.5. We provide the main differences to the edge-embedding of the 4-way EUG construction detailed in §3.1.

***k*-way Block Edge-Embedding.** In this setting, our main block is a programmable block B of size x 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 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, respectively. Values $k, \ldots, 2k-1$ in out denote the recursion point targets $r_1^1, \ldots, r_1^k$ (cf. §3.1). We formalize the setting of in and out in Eqs. 10–14. We denote with $\mathcal{P}$ the set of all paths in B, and the $\mathcal{P}_B \subseteq P$ the paths that get edge-embedded in B.

$$\forall i \in \{1, \ldots, k\} : in_i \neq 0 \rightarrow (r_0^i, p_{in_i}) \in \mathcal{P}_B, \tag{10}$$

$$out_i \neq 0 \wedge out_i < k \rightarrow (p_i, p_{1+out_i}) \in \mathcal{P}_B \wedge \eta^U(p_i) < \eta^U(p_{1+out_i}) \tag{11}$$

$$out_i > k-1 \rightarrow (p_i, r_1^{i-k+1}) \in \mathcal{P}_B. \tag{12}$$

$$\forall in_i, in_j \in in : i \neq j \rightarrow in_i = 0 \vee in_i \neq in_j. \tag{13}$$

$$\forall out_i, out_j \in out : i \neq j \rightarrow out_i = 0 \vee out_i \neq out_j. \tag{14}$$

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

1. Splitting $G \in \Gamma_2(n)$ in two $\Gamma_1(n)$ graphs G_1 and G_2: Similarly as in §3.1, 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. §3.1) 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 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:

$$\forall i, j \in \{1, \ldots, k\} : i \neq j \to E_i \cap E_j = \emptyset. \tag{15}$$

$$\forall e \in E : \exists i \in \{1, \ldots, k\} : e \in E_i. \tag{16}$$

$$\forall i \in \{1, \ldots, k\} : \forall e = (v_1, v_2) \in E_i : \\ \neg \exists e' = (v_3, v_4) \in E_i : v_2 = v_4 \vee v_1 = v_3. \tag{17}$$

According to Kőnig's theorem [Kő31, LP09], $\Gamma_k(n)$ graphs can always be k-colored efficiently with a dedicated algorithm (cf. Appendix A). The rest of the supergraph construction and the way it is used for edge-embedding is the same as for the 4-way EUG construction as described in §3.1.

***k*-way Edge Embedding Algorithm.** The edge-embedding algorithm is the same as shown in Listing 1, after replacing every 4 with k.

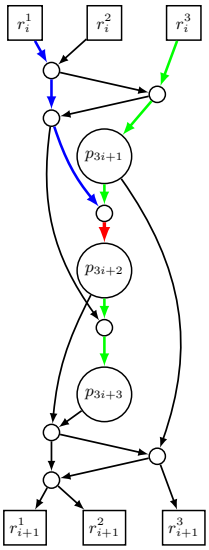
4 New Universal Circuit Constructions

Here, we describe our ideas for novel, potentially more efficient, UC constructions. Firstly, in §4.1, we describe modular building blocks for a *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 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 UC constructions. Secondly, in §4.2, we propose a *hybrid UC construction*, utilizing both Valiant's 2-way and 4-way UC constructions 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.

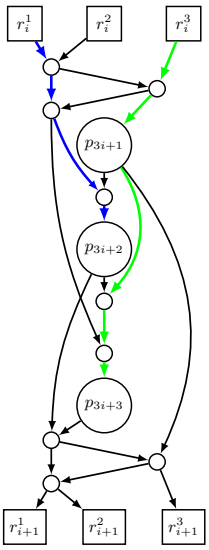
4.1 3-way Universal Circuit Construction

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 construction. 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., Fig. 5a for the 4-way UC). Once it is determined, the size of the UC is $\mathrm{size}(U_n^{(k)}(\Gamma_2)) = 2 \cdot \mathrm{size}(U_n^{(k)}(\Gamma_1)) \approx 2 \cdot \frac{\mathrm{size}(B_k)}{k} n \log_k n = 2 \cdot \frac{\mathrm{size}(B_k)}{k \log_2(k)} n \log_2 n$. The modular block consists of two permutation networks $P(k)$, an EUG $U_k(\Gamma_1)$, and $(k-1)$ Y-switching blocks (cf. §2.5, [LMS16]).

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 §2.2). An optimal permutation network $P(3)$ that achieves the lower bound has 3 nodes as well. This implies that $\mathrm{size}(B_3) = 2 \cdot P(3) + \mathrm{size}(U_3(\Gamma_1)) + (3-1) = 11$. Then, the size of the UC becomes $\approx 2 \cdot \frac{11}{3 \log_2 3} n \log_2 n \approx 4.627 n \log_2 n$, which means an asymptotically by around 2.5% smaller size than that of the 4-way UC.



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



(b) Body Block with our generic $U_3(\Gamma_1)$

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

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 construction. 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 construction, it cannot directly be applied. A generic $U_3(\Gamma_1)$ that can embed $(1, 3)$ without going through p_2 as before has an additional Y-switching block.

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}. Then, it needs to go through pole p_{3i+2}, which means that the edge going in to p_{3i+2} is used by this path. However, there might 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. This cannot be directed to p_{3i+2} anymore as shown in Fig. 6a. Therefore, in the 3-way body block construction, it does not suffice to use Valiant's optimized $U_3(\Gamma_1)$ [Val76].

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}. This results in $\text{size}(B_3) = 2 \cdot P(3) + \text{size}(U_3(\Gamma_1)) + (3-1) = 12$, which implies that the asymptotic size of the UC is $\approx 2 \cdot \frac{12}{3 \log_2 3} n \log_2 n \approx 5.047 n \log_2 n$. Unfortunately, this is worse than the asymptotic size of the 2-way construction, and we therefore conclude that the asymptotically most efficient known UC construction is Valiant's 4-way UC construction.

Listing 2: Hybrid construction algorithm

```
1  procedure hybrid (p_1, ..., p_n, K = {2,4})
2   Let size(U_{n'}^{hybrid}(Γ_1)) be the function calculating the size of the smaller hybrid constructions
        ↪ with size n' ≤ n
3   for all k ∈ K do // Number of poles in the last block for all k
4    if n | k do
5        m_k ← k
6    else
7        m_k ← n mod k
8    end if
9    s_k ← size(Head_k(k)) + (⌈n/k⌉ − 3) · size(Body_k(k)) + size(Body_k(r_k)) + size(Tail_k(m_k)) +
        ↪ m_2 · size(size(U^{hybrid}_{⌈n/2 − 1⌉}(Γ_1))) + ((k − m_k) · size(size(U^{hybrid}_{⌊n/k − 1⌋}(Γ_1)))
10  end for
11  s_i ← min(s_k : k ∈ K) // Choose the better construction
12  Create skeleton for i−way construction with n poles
13  call hybrid(r_1^1, ..., r^1_{⌈n/i − 1⌉}, K), ..., hybrid(r_1^{m_i}, ..., r^{m_i}_{⌈n/i − 1⌉}, K)
14  if (i − m_i) > 0 do
15   call hybrid(r_1^{m_i}, ..., r^{m_i}_{⌊n/i − 1⌋}, K), ..., hybrid(r_1^i, ..., r^i_{⌊n/i − 1⌋}, K)
16  end if
17 end procedure
```

4.2 Hybrid Universal Circuit Construction

In this section, we detail our hybrid UC that minimizes its size based on Valiant's 2-way and 4-way UCs, which are asymptotically the smallest UCs to date. Given the size of the input circuit $C^k_{u,v}$, i.e., $n = u + k + v$, 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 $(\text{size}(U_n^{\text{hybrid}}(\Gamma_1)))$ of the hybrid construction for sizes smaller than n. We give details on how it is computed in §5. 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 UC constructions as parameter K ($K = \{2, 4\}$ in our example), it minimizes the concrete size of the resulting UC.

5 Size and Depth of UC Constructions

Lipmaa et al.'s k-way UC construction is depicted in a modular manner in [LMS16, Fig. 12] and discussed briefly in §2.5 and Fig. 4. They show that a k-way body block consists 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 (Table 2) and depths (Table 3) of the k-way EUG and give an estimate for the leading constant for Lipmaa et al.'s EUG construction with size $\mathcal{O}(n \log_2 n)$ and depth $\mathcal{O}(n)$, for $k \in \{2, \ldots, 8\}$. We conclude that the best asymptotic size is achieved by Valiant's 4-way UC. This result does not exclude the possibility for a more efficient UC in general, but it shows that the most efficient UC using Lipmaa et al.'s k-way UC from [LMS16] is the 4-way UC. Two k-way EUGs for $\Gamma_1(n)$ graphs build up an EUG for $\Gamma_2(n)$ graphs as described in §2.1. Therefore, the leading constant for the size of the UC is twice that of the EUG $U_n^{(k)}(\Gamma_1)$, which is summarized in the same table.

k	$U_k(\Gamma_1)$	$U^{\text{KS08}}(k)$	$P^{\text{l}}(k)$	$P^{\text{W}}(k)$	B_k^W	$U_n^{(k)}(\Gamma_1)$ $(\cdot n \log_2 n)$	UC $(\cdot n \log_2 n)$
2	**2**	2	1	**1**	5	2.5	5
3	**4**	6	3	**3**	12	≈ 2.524	≈ 5.047
4	**6**	7	5	**5**	19	2.375	4.75
5	**10**	11	7	**8**	30	≈ 2.584	≈ 5.168
6	**13**	14	10	**11**	40	≈ 2.579	≈ 5.158
7	**19**	19	13	**14**	53	≈ 2.697	≈ 5.394
8	23	**21**	16	**17**	62	≈ 2.583	≈ 5.167

Table 2: The leading factors of the asymptotic $\mathcal{O}(n \log_2 n)$ **size** for k-way edge-universal graphs ($U_n^{(k)}(\Gamma_1)$) and universal circuits (UC) for $k \in \{2, \ldots, 8\}$. n denotes the size of the input $\Gamma_2(n)$ circuit, $U_k(\Gamma_1)$ the size of Valiant's edge-universal graph with k poles, $U^{\text{KS08}}(k)$ the size of the UC of [KS08b], $P^{\text{l}}(k)$ the lower bound for the size of a permutation network for k nodes, and $P^{\text{W}}(k)$ the size of Waksman's permutation network [Wak68]. B_k^W is the size of the body block.

5.1 Asymptotic Size and Depth of k-Way UC Constructions

We review the sizes of the building blocks of a k-way body block, i.e., the size of an EUG $U_k(\Gamma_1)$ for k, and the size of a permutation network $P(k)$ with k inputs and outputs, as well as the size of the resulting UCs.

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 connected poles, for $k = 3$ we discussed in §4.1 that an additional node is necessary. For $k \in \{4, 5, 6\}$ the sizes are $\{6, 10, 13\}$, as shown in [KS16, Fig. 1] (note that the nodes noted as empty circles disappear in the UC and therefore are not counted here). For $k = 7$ and $k = 8$, we observe that Valiant's 2-way UC construction results in a better size than that of the 4-way UC construction 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 $U^{\text{KS08}}(k)$ are built from two smaller $U^{\text{KS08}}(\frac{k}{2})$, a $P(\frac{k}{2})$ and $\frac{k}{2}$ parallel Y switches. 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].

Permutation Networks.

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 $P^{\text{l}}(k)$. The permutation network with the smallest size is Waksman's permutation network $P^{\text{W}}(k)$ [Wak68, BD02]. For $k \in \{2, 3, 4\}$ its size reaches the lower bound, but for larger k values, his permutation network utilizes additional nodes. Since these are the smallest existing permutation networks, we use these when calculating the size of the UC. Even with the lower bound $P^{\text{l}}(k)$, for $k \in \{5, 6, 7, 8\}$ we would have the respective leading terms $\{4.824, 4.900, 5.190, 5\}$, which are larger than 4.75 for $k = 4$.

k	$U_k(\Gamma_1)$	$U^{\text{KS08}}(k)$	$P^{\text{l}}(k)$	$P^{\text{W}}(k)$	B_k^W	$U_n^{(k)}(\Gamma_1)$ $(\cdot n)$	UC $(\cdot n)$
2	**2**	2	1	**1**	6	3	3
3	**4**	5	3	**3**	13	≈ 4.333	≈ 4.333
4	**5**	6	3	**3**	15	3.75	3.75
5	**7**	9	4	**5**	22	4.4	4.4
6	**10**	12	4	**5**	26	≈ 4.333	≈ 4.333
7	16	**14**	4	**5**	31	≈ 4.429	≈ 4.429
8	19	**16**	4	**5**	34	4.25	4.25

Table 3: The leading factors of the asymptotic $\mathcal{O}(n)$ **depth** for k-way edge-universal graphs ($U_n^{(k)}(\Gamma_1)$) and universal circuits (UC) for $k \in \{2, \ldots, 8\}$. n denotes the size of the input $\Gamma_2(n)$ circuit, $U_k(\Gamma_1)$ the depth of Valiant's edge-universal graph with k poles, $U^{\text{KS08}}(k)$ the depth of the UC of [KS08b], $P^{\text{l}}(k)$ the lower bound for the depth of a permutation network for k nodes, and $P^{\text{W}}(k)$ the depth of Waksman's permutation network [Wak68]. B_k^W is the depth of the k-way body block.

Depth: The depth of a permutation network has lower bound $\log_2(k) + 1$, 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 $P^{\text{l}}(k)$ in Table 3. Waksman's permutation network achieves the lower bound when $k \in \{2, 3, 4\}$, but utilizes additional nodes for larger k values.

Body Blocks. A body block B_k^W is built of $(k-1)$ Y-switching blocks, an EUG for k nodes, and two permutation networks [LMS16] (cf. Fig. 4).

Size: The size of the body block with Waksman's permutation network B_k^W is the sum of the sizes of its building blocks, i.e., $\text{size}(B_k^W) = \min\big(\text{size}(U_k(\Gamma_1)), \text{size}(U^{\text{KS08}}(k))\big) + 2 \cdot \text{size}(P^W(k)) + k - 1$.

Depth: The depth of B_k^W 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 $\text{depth}(B_k^W) = \min\big(\text{depth}(U_k(\Gamma_1)), \text{depth}(U^{\text{KS08}}(k))\big) + 2 \cdot \text{depth}(P^W(k)) + k - 1 + 1$.

Edge-Universal Graphs and Universal Circuits with n Poles.

Size: The asymptotic size of EUG $U_n^{(k)}(\Gamma_1)$ is determined as $\text{size}(U_n^{(k)}(\Gamma_1)) = \frac{\text{size}(B_k^W)}{k \log_2 k} n \log_2 n$ and the leading factor for a UC is twice this number.

Depth: The depths of the EUG and of the UC depend only on the depth of the outest 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 $\frac{\text{depth}(B_k^W)}{k}$.

5.2 Concrete Size and Depth of UC Constructions

Both the size and depth of Lipmaa et al.'s k-way universal circuits depends on the size and depth of their building blocks [LMS16]. More concretely, finding either better edge-universal graphs for small number of nodes or optimal permutation networks could improve the sizes and depths of these UCs. Lipmaa et al. calculated the optimal k value for minimizing the size of a k-way UC to be 3.147 [LMS16].

Table 2 shows that the smallest sizes are achieved by the 4-way, followed by the 2-way UCs. The 3-way UC, as mentioned in §4.1, is less efficient due to the additional node in $U_3(\Gamma_1)$. We observe that the sizes grow with increasing k values due to the large permutation networks and EUGs. The depth is minimal for the 2-way, followed by the 4-way UCs as shown in Table 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. Kiss and Schneider considered this in their 2-way UC 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 defined as

$$m_k := \begin{cases} n \mod k & \text{if } k \nmid n, \\ k & \text{if } k \mid n. \end{cases} \tag{18}$$

Then, given the designed Head, Body and Tail blocks 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 shown in Eq. 19.[4]

$$\begin{aligned} \text{size}(U_n^{(k)}(\Gamma_1)) = \text{size}(\text{Head}(k)) + \left(\left\lceil \frac{n}{k} \right\rceil - 3\right) \cdot \text{size}(\text{Body}(k)) + \\ \text{size}(\text{Body}(m_k)) + \text{size}(\text{Tail}(m_k)) + \\ m_k \cdot \text{size}\left(U^{(k)}_{\lceil \frac{n}{k}-1 \rceil}(\Gamma_1)\right) + (k - m_k) \cdot \text{size}\left(U^{(k)}_{\lfloor \frac{n}{k}-1 \rfloor}(\Gamma_1)\right). \end{aligned} \tag{19}$$

The depth of a k-way universal circuit also requires the above defined m_k, the Head, Tail and Body blocks, but does not rely on the subgraphs. Therefore, it can be calculated using the closed formula shown in Eq. 20.

$$\begin{aligned} \text{depth}(U_n^{(k)}(\Gamma_1)) = \text{depth}(\text{Head}(k)) + \left(\left\lceil \frac{n}{k} \right\rceil - 3\right) \cdot \text{depth}(\text{Body}(k)) + \\ \text{depth}(\text{Body}(m_k)) + \text{depth}(\text{Tail}(m_k)). \end{aligned} \tag{20}$$

Concrete Size and Depth of our Hybrid UC. We provide a hybrid UC in §4.2 for minimizing the size of the resulting UC. This construction chooses at each step the skeleton that results in the smallest size and therefore, we provide the recursive algorithm for determining its size in Eq. 21. Its depth is the depth of the outest skeleton. $\text{size}(\text{Head}_k(i))$, $\text{size}(\text{Tail}_k(i))$ and $\text{size}(\text{Body}_k(i))$ are the values from Table 4 for $k = 2$ and $k = 4$. The size of the hybrid UC is minimized as

$$\begin{aligned} \text{size}(U_n^{\text{hybrid}}(\Gamma_1)) = \min \Bigg(\text{size}(\text{Head}_k(k)) + \left(\left\lceil \frac{n}{k} \right\rceil - 3\right) \cdot \text{size}(\text{Body}_k(k)) + \\ \text{size}(\text{Body}_k(m_k)) + \text{size}(\text{Tail}_k(m_k)) + m_k \cdot \text{size}\left(U^{\text{hybrid}}_{\lceil \frac{n}{k}-1 \rceil}(\Gamma_1)\right) + \\ (k - m_k) \cdot \text{size}\left(U^{\text{hybrid}}_{\lfloor \frac{n}{k}-1 \rfloor}(\Gamma_1)\right);\ \ k \in \{2, 4\} \Bigg), \end{aligned} \tag{21}$$

which can be computed using a dynamic programming algorithm.

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

Block	Head				Body				Tail			
k\Poles	4	3	2	1	4	3	2	1	4	3	2	1
Fig.	-	-	3c	-	-	-	3a	3b	-	-	3d	3e
2-way size	-	-	4	-	-	-	5	5	-	-	4	1
2-way depth	-	-	4	-	-	-	6	6	-	-	4	1
Fig.	5e	5g	5h	5i	5a	5b	5c	5d	5f	5g	5h	5i
4-way size	14	14	13	12	19	19	18	17	14	9	4	1
4-way depth	11	11	10	10	15	15	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, 5).

Improvement of 4-way Construction. The bottom (blue) line in Fig. 7 shows the concrete improvement in percentage of the 4-way UC construction over the 2-way UC construction up to ten million nodes in the simulated input circuit. From the asymptotic leading factors in Table 2, we expect an improvement of up to $1 - \frac{4.75}{5} = 5\%$. For the smallest n values ($n \leq 15$), the 2-way UC is up to 33.3% better than the 4-way UC. However, from $n = 212$ on, the 4-way UC construction is better, except for some short intervals as shown in Fig. 7 (the difference in these intervals, however, is at most 3.45%). From here on, the 4-way UC is on average 3.12% better in our experiments, where the biggest improvement is 4.48%. Moreover, from $n = 10\,885$ on, the 4-way UC always outperforms the 2-way UC.

Improvement of Hybrid Construction. The improvement achieved by our hybrid construction (cf. §4.2) is depicted in the same Fig. 7, as the top (green) line. For some n values the hybrid UC achieves the same size as the 2- or 4-way UCs, but due to its nature, it is never worse. This means that the improvement of our hybrid UC is always nonnegative, and greater than or equal to the improvement achieved by the 4-way UC. Moreover, in most cases the hybrid UC results in better sizes than any of the other two constructions: this means that some subgraphs are created for an n for which the 2-way UC is smaller, and therefore the 2-way recursive structure is utilized. The overall improvement for all n values is on average 3.65% and at most 4.48% over the 2-way UC construction.

6 Implementation and Evaluation

The first implementation of Valiant's 2-way UC, along with a toolchain for PFE (cf. §1.1) was given in [KS16]. The 4-way UC has smaller asymptotic size $\sim 4.75n \log_2 n$, but has not been implemented before due to its more complicated structure and embedding algorithm.

In this work, we improve the implementation of the open-source framework of [KS16] by using the 4-way UC construction that can directly be applied in the PFE framework. Our improved implementation is available at `http://encrypto.de/code/UC`. Firstly, the functionality is translated to a Boolean circuit using the Fairplay compiler [MNPS04, BNP08]. This is then transformed into a circuit in $\Gamma_2(n)$, i.e., with at most two incoming and outgoing wires for each gate, input and output. This is done in a preprocessing step of the framework in [KS16]. The input circuit description of our UC implementation is the same as that of the UC compiler of [KS16], and we also adapt our output UC format to that of [KS16] that includes the gate types described in §2.2. This format is compatible with the ABY framework [DSZ15] for secure function evaluation.

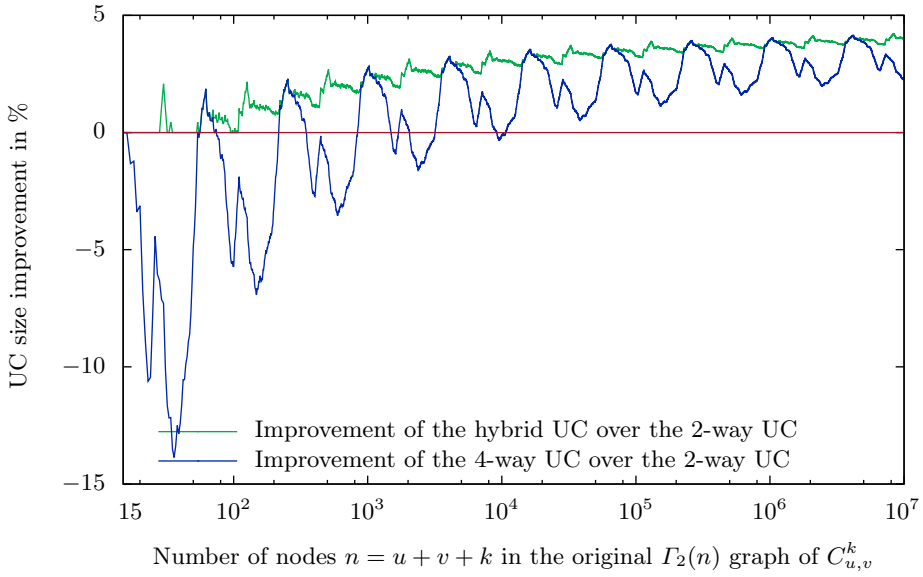


Fig. 7: Improvement of our hybrid and Valiant's 4-way UC over Valiant's 2-way UC for $15 \leq n \leq 10^7$ with logarithmic x axis.

We discuss our implementation of Valiant's 4-way UC in §6.1 and give experimental results in §6.2. We describe future work in §6.3.

6.1 Our 4-Way Universal Circuit Implementation

The architecture of our UC implementation is the same as that of [KS16], and therefore, we describe our UC design based on the steps described in [KS16, Fig. 6]. Our implementation gets as input a circuit with u inputs, v outputs and k gates, and outputs a 4-way UC with size $n = u + k + v$, as well as the programming p_f corresponding to the input circuit (cf. §1).

Transforming circuit $C^k_{u,v}$ into $\Gamma_2(u+k+v)$ graph G. As a first step, we transform the circuit $C^k_{u,v}$ into a $\Gamma_2(n)$ graph $G = (V, E)$ with $n = u+k+v$ (cf. §2.1). Then, we define a topological order η^G on the nodes of G s.t. 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 $u + k + 1 \leq \eta^G(v_j) \leq u + k + v$.

Creating an EUG $U^{(4)}_n(\Gamma_2)$ for $\Gamma_2(n)$ graphs. An EUG $U^{(4)}_n(\Gamma_2)$ is constructed by creating two instances of $U^{(4)}_n(\Gamma_1)$ as shown in §2.2. The two instances get merged to $U^{(4)}_n(\Gamma_2)$ 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). We create the EUGs with Valiant's 4-way EUG [Val76] with our optimized blocks from §3.1 (cf. Fig. 5).

Programming $U^{(4)}_n(\Gamma_2)$ to compute $C^k_{u,v}$. We edge-embed graph G into $U^{(4)}_n(\Gamma_2)$ as described in §3.1. [KS16] use their supergraph construction to define the paths between the poles uniquely for Valiant's 2-way EUG. We modify this supergraph as described in §3.1 for Valiant's 4-way EUG and perform the edge-embedding as described in Listing 1. The programming bits of the nodes are set

Circuit	n	Circuit size (#AND gates)			UC generation (ms)	
	$u+v+k$	2-way UC [KS16]	Our 4-way UC	Our Hybrid UC	2-way UC [KS16]	Our 4-way UC
`AES-non-exp`	46 847	$2.96 \cdot 10^6$	$\mathbf{2.93 \cdot 10^6}$	$2.86 \cdot 10^6$	9 008.9	10 325.8
`AES-exp`	38 518	$2.39 \cdot 10^6$	$\mathbf{2.38 \cdot 10^6}$	$2.31 \cdot 10^6$	6 961.7	8 361.3
`DES-non-exp`	31 946	$1.96 \cdot 10^6$	$\mathbf{1.92 \cdot 10^6}$	$1.89 \cdot 10^6$	5 563.8	6 599.5
`DES-exp`	32 207	$1.98 \cdot 10^6$	$\mathbf{1.94 \cdot 10^6}$	$1.90 \cdot 10^6$	5 654.0	6 765.0
`md5`	66 497	$4.42 \cdot 10^6$	$\mathbf{4.26 \cdot 10^6}$	$4.26 \cdot 10^6$	14 805.5	14 897.8
`sha-256`	201 206	$1.49 \cdot 10^7$	$\mathbf{1.46 \cdot 10^7}$	$1.44 \cdot 10^7$	81 889.1	57 439.0
`add_32`	342	$9.58 \cdot 10^3$	$\mathbf{9.55 \cdot 10^3}$	$9.44 \cdot 10^3$	29.6	35.3
`add_64`	674	$\mathbf{2.21 \cdot 10^4}$	$2.27 \cdot 10^4$	$2.17 \cdot 10^4$	53.9	89.6
`comp_32`	216	$\mathbf{5.53 \cdot 10^3}$	$5.54 \cdot 10^3$	$5.49 \cdot 10^3$	17.7	21.2
`mult_32x32`	12 202	$6.54 \cdot 10^5$	$\mathbf{6.50 \cdot 10^5}$	$6.35 \cdot 10^5$	1 639.2	2 177.1
`Branching_18`	200	$\mathbf{4.92 \cdot 10^3}$	$5.07 \cdot 10^3$	$4.88 \cdot 10^3$	21.0	24.2
`CreditChecking`	82	$\mathbf{1.50 \cdot 10^3}$	$1.51 \cdot 10^3$	$1.49 \cdot 10^3$	3.1	12.7
`MobileCode`	160	$\mathbf{3.65 \cdot 10^3}$	$3.88 \cdot 10^3$	$3.61 \cdot 10^3$	10.6	29.0

Table 5: Comparison of the sizes of the UCs (2-way, 4-way, and hybrid) for sample circuits from [TS15]. Bold numbers denote if the 2-way or the 4-way UC is smaller; the smallest size is always achieved by our hybrid UC. The UC generation time is given for both implemented UCs.

during the edge-embedding process along the paths between the poles. The block edge-embedding is done by analyzing the possible input values and defining the valid paths as described in §3.1.

Outputting a universal circuit with its programming. As a final step, EUG $U_n^{(4)}(\Gamma_2)$ is topologically ordered and output in the UC format of [KS16]. The programming bits p_f defined by the embedding are also output in a separate file based on the topological order.

6.2 Our Experimental Results

In order to show the improvement of our method, we ran experiments on a Desktop PC, equipped with an Intel Haswell i7-4770K CPU with 3.5 GHz and 16 GB RAM, and provide our results in Table 5. To compare with the runtime of the UC compiler of [KS16], we ran the same experiments on the same platform using their 2-way UC implementation.

As [KS16], we use a set of real-life circuits from [TS15] for our benchmarks, and compare the sizes of the resulting circuits and the generation and embedding runtimes. We can see that from the 2-way and 4-way UC constructions, the 4-way UC, as expected, is always smaller for large circuits than the 2-way UC. However, it is sometimes better even for small circuits, e.g., for 32-bit addition with $n = 342$. The hybrid construction always provides the smallest UC for our example circuits.

In the last two columns, we report the runtime of the UC compiler of [KS16] and our 4-way UC implementation for generating and programming the universal circuit corresponding to the example circuits. Table 5 shows that the differences in runtime are not significant, and due to its more complicated structure, the 4-way UC takes more time to generate and program in general. However, we can see from the largest example, i.e., SHA-256 with more than 200 000 nodes in the input circuit, that asymptotically, the 4-way UC results in a runtime improvement as well, as less nodes need to be programmed.

6.3 Future Work: Hybrid Universal Circuit Implementation

We leave implementing our hybrid UC as future work. Note that adapting the edge-embedding algorithm is straightforward with our methods: the block edge-embedding is the same, depending on k and the block type at each step (cf. Listing 2). The recursion point edge-embedding can also be adapted easily: First, we need to split the input $\Gamma_2(n)$ graph into two $\Gamma_1(n)$ graphs as before. Then, if $k = 2$ in the next recursion step, we do the splitting and merging of [KS16]. If $k = 4$, we include the preprocessing step. We continue recursively until we reach the recursion base.

Though it can be implemented straightforwardly, the UC generation and programming might take slightly longer for the hybrid UC. One reason for this is that for any value of n, to be able to decide on which k to use at each step, we need the size of the smaller hybrid construction. Therefore, we need to run an algorithm (cf. Eq. 21) that generates the sizes of the hybrid UC up to n nodes before we can start building our UC. However, this can be pre-computed efficiently using a dynamic programming algorithm and hence is a one-time expense.

Acknowledgements. This work has been co-funded by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP and by the DFG as part of project E3 within CROSSING. We thank the reviewers of ASIACRYPT'17 for their helpful comments.

References

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

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. http://ia.cr/2014/772.

BBKL17. O. Bicer, M. A. Bingol, 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. http://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.

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.

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.

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

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: http://encrypto.de/code/ABY.

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.

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.

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*, LNCS. Springer, 2017. Code: http://encrypto.de/code/UC.

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.

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

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.

KR11. S. Kamara and M. Raykova. Secure outsourced computation in a multi-tenant cloud. In *IBM Workshop on Cryptography and Security in Clouds*, 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: http://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. Code: http://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. http://ia.cr/2016/017.

LP09. 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.

MNPS04. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - 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 *ACM Conference on Electronic Commerce (EC'99)*, pages 129–139. ACM, 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.

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.

Sad15. S. S. Sadeghian. *New Techniques for Private Function Evaluation*. PhD thesis, University of Calgary, December 2015.

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

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

SS08. A.-R. Sadeghi and T. Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *ICISC'08*, volume 5461 of *LNCS*, pages 336–353. Springer, 2008.

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

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.

A Algorithm for k-Coloring Graphs with Fanin and Fanout k

Kiss and Schneider in [KS16] show that the first step for 2-coloring a graph with fanin and fanout 2 is to transform the graph $G = (V, E)$ with n nodes ($V = \{v_1, \ldots, v_n\}$) to a bipartite graph $\overline{G} = (\overline{V}, \overline{E})$ with $\overline{V} = \{w_1, \ldots, w_n, z_1, \ldots, z_n\}$. The edges are set such that $(w_i, z_j) \in \overline{E}$ if and only if $(n_i, n_j) \in E$. We can easily see that the fanin and fanout of the resulting bipartite graph is also 2. 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.

Kőnig's theorem was used in [KS16, LMS16] to provide the 2-coloring algorithm for a graph with fanin and fanout 2. In its proposed form, however, Kőnig's theorem [Kő31, LP09] applies also for k-coloring any graph with at most k incoming and outgoing edges for each of its nodes. We review this theorem and the corresponding algorithm which can be used for embedding any graph into any k-way UC as described in §3.2.

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 and algorithm. 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, then it 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 $\overline{G}$.

B Notations

UC	Universal circuit, a circuit that can be programmed to evaluate any circuit up to a given size.
f	Function to be privately evaluated using a universal circuit.
p_f	Programming 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.
k	Number of gates in simulated Boolean circuit.
$C^k_{u,v}$	The Boolean circuit that describes f with fanin and fanout 2.
G	The graph of $C^k_{u,v}$ where every input, output and gate is represented with a node and every wire is represented with an edge.
n	Size of the simulated circuit $C^k_{u,v}$ and its graph G, sum of its u inputs, v outputs and k gates.
$G = (V, E)$	G graph with node set $V = \{1, \ldots, n\}$ and set of edges $E \subseteq V \times V$.
$\Gamma_d(n)$	The set of all graphs with fanin and fanout d and n nodes.
$U_n(\Gamma_d)$	Edge-universal graph for $\Gamma_d(n)$ graphs, used generically and for the recursion base EUGs.
$U^{(k)}_n(\Gamma_d)$	k-way edge-universal graph for $\Gamma_d(n)$ graphs.
$U^{\text{hybrid}}_n(\Gamma_d)$	Hybrid edge-universal graph for $\Gamma_d(n)$ graphs.
p_i	Distinguished nodes in $U_n(\Gamma_d)$, called poles, with fanin and fanout d.
P	Set of all poles in $U_n(\Gamma_d)$.
UG	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.
f_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.
f_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.
$P^l(k)$	Lower bound on the size of the permutation network for k nodes.
$P^W(k)$	Size of the Waksman's permutation network [Wak68] for k nodes.
$U^{\text{KS08}}(k)$	The size of the UC of [KS08b].