Rajesh Krishnan and Ravi Sundaram*

Secure and scalable match: overcoming the universal circuit bottleneck using group programs

Abstract: Confidential Content-Based Publish/Subscribe (C-CBPS) is an interaction model that allows parties to exchange content while protecting their security and privacy interests. In this paper we advance the state of the art in C-CBPS by showing how all predicate circuits in NC^1 (logarithmic-depth, bounded fan-in) can be confidentially computed by a broker while guaranteeing perfect information-theoretic security. Previous work could handle only strictly shallower circuits (e.g. those with depth $O(\sqrt{\log_2 n})$). We present three protocols—UGP-Match, FSGP-Match and OFSGP-Match—based on 2-decomposable randomized encodings of group programs for circuits in NC^1. UGP-Match is conceptually simple and has a clean proof of correctness but its running time is a polynomial with a high exponent and hence impractical. FSGP-Match uses a "fixed structure" construction that reduces the exponent drastically and achieves efficiency and scalability. OFSGP-Match optimizes the group programs further to shave off a linear factor.

Keywords: secure computation, publish-subscribe, universal circuit, group program, randomized encoding

DOI 10.1515/popets-2015-0013
Received 2014-11-15; revised 2015-03-02; accepted 2015-05-15.

1 Introduction

1.1 Motivation

Publish/subscribe systems are an efficient means of routing relevant information from publishers or content generators to subscribers or consumers. The efficiency of publish/subscribe models comes from the fact that subscribers typically receive only a subset of the total messages published. The process of selecting messages for reception and processing is called filtering. There are two common forms of filtering: topic-based and content-based. In this paper we focus on Content-Based Publish/Subscribe systems (CBPS) where messages are only delivered to a subscriber if the attributes or metadata of those messages match predicates defined by the subscriber. The subscriber is responsible for specifying his preferences as a predicate over the attributes of the content produced by the publisher. For the purposes of this paper we will assume that the predicate is expressed as a circuit and we will use the terms predicate and circuit interchangeably. There is a third party, the broker, who is responsible for matching the subscriber's predicate to the metadata produced by the publisher and, in case of a match, forwarding the associated data to the subscriber, see Figure 1 for the basic interaction pattern. The loose coupling between subscribers and publishers enabled by the broker allows for greater scalability. CBPS is an incredibly useful means of disseminating information and can be viewed as an abstraction for a variety of different applications ranging from forwarding of e-mail to distributed event management to query/response systems built on top of databases.

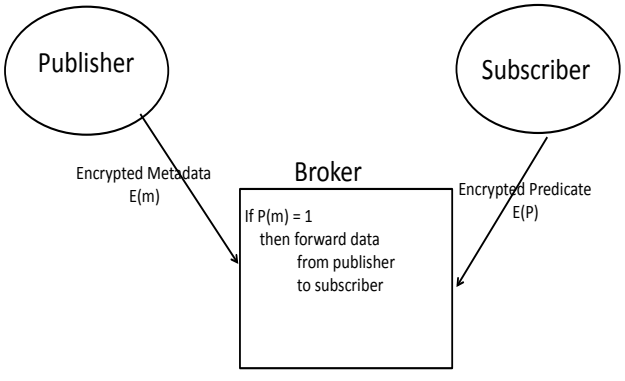

Fig. 1. The C-CBPS interaction model

Scalability of CBPS systems and the distributed coordination of a mesh of brokers are natural concerns.

Rajesh Krishnan: Cosocket LLC, E-mail: krash@cosocket.com
***Corresponding Author: Ravi Sundaram:** Northeastern University, E-mail: koods@ccs.neu.edu

But, in recent times, with the proliferation of online social networks and new forms of social media an even more pressing concern has come into focus, namely confidentiality. *Publisher confidentiality* refers to the notion that the publisher would like to keep his content secure from the broker, e.g., the stock exchange would like to keep ticker/price information private to prevent reselling. *Subscriber confidentiality* is the notion that subscribers would like to keep their preferences private, e.g. a hedge fund would not wish to reveal their interest in a particular stock. The widespread development of web and mobile apps has created a proliferation of third-parties involved in the business of handling user preferences and routing content, e.g. iPhone and Android apps, Facebook and Twitter apps. But even in earlier times, there has always been the need for preserving the privacy of both the publisher and the subscriber. For example, a database server must not learn what information was requested by a client, and yet have the assurance that the client was authorized to have the information that was sent; a mail relay must be able to forward the relevant emails without learning the contents of the email or the subscribers of a mail-list. This motivates the need for a Confidential Content-Based Publish/Subscribe (C-CBPS) scheme such as the one illustrated in Figure 1, where the publisher's metadata and the subscriber's predicate are encrypted and the broker performs the matching operation without decryption.

1.2 Our Results

The problem of securing the privacy of publishers and subscribers is recognized as a challenging and important problem. Practical C-CBPS systems can handle the confidentiality of exact matches and some minor variants [30]. Sophisticated schemes handling more expressive subscriber predicates are too slow in practice. It has been hard to attain practicality and expressivity.

Reusing the technology of group programs [6] in the context of decomposable randomized encodings [1] we build on [13] to obtain both - a theoretical advance on the state of the art, and a fast protocol with real-world potential. We have two main contributions in this paper.

– We present an information-theoretically secure protocol, UGP-Match employing *universal group programs* to match any predicate in NC^1, i.e. logarithmic-depth bounded fan-in circuits [4, 6]. UGP-Match demonstrates the theoretical possibility of attaining NC^1 but is not practical.
– We then show how a "fixed structure" construction gets us substantially closer to creating a practical and real-world protocol, FSGP-Match, to achieve fast and secure matching of any predicate in NC^1. FSGP-Match is identical to UGP-Match in security guarantees but is much faster. Finally, we gain additional efficiency by using an innovative optimization to contract group programs further in our fastest protocol, OFSGP-Match.

Our schemes provide us with a high level of expressivity (NC^1) and in the strongest model of security (information-theoretic). The above C-CBPS protocols can be converted to use shared seeds and pseudorandom number generators (PRNGs) rather than shared randomness. In case PRNGs are used, the resulting protocols are easily seen to be secure in the computational setting based on the unpredictability of the PRNGs.

We have built prototypes of our protocols and characterized its performance.

The focus of this paper is on achieving a secure match algorithm that is scalable. Therefore we concentrated on bounded depth predicates, i.e. predicates in NC^1. But we also have additional asymptotic and complexity-theoretic improvements which are not practical. For completeness we mention them here but we do not present the proofs or constructions in this paper. One, using *universal branching programs* we can give an information-theoretically secure (C-CBPS) protocol for matching any predicate in NL(nondeterministic logspace). Second, using randomized encodings we can give a *computationally secure* (C-CBPS) protocol to match any predicate in P (polynomial time).

1.3 Related Work

Several practical CBPS systems have been built for supporting a variety of distributed applications. Siena [9] is one of the most well known; Gryphon [5] and Scribe [12] are others. Work on the security aspects, namely C-CBPS in the systems community is less than a decade old. A C-CBPS system supporting only equality matches is presented in [32] and a system supporting extensions to inequality and range matches is presented in [30]. Both these systems are in the setting of computational security. However, neither of these systems satisfy our confidentiality model since they allow the broker to see that the encrypted predicates from two different subscribers are identical.

As mentioned earlier, C-CBPS is a special case of SMC and there is a large body of literature on SMC within the cryptographic community [15, 16, 22]. The special case of 2-party SMC, known as Secure Function Evaluation (SFE), is an important subcase [24] that is (different from but) closely related to the C-CBPS model. Research in SMC started nearly 3 decades ago with the path-breaking work of Yao, [40], and is broadly divided into two classes of protocols – those that are computationally secure (i.e. conditioned on certain complexity-theoretic assumptions) and those that are information-theoretically (or unconditionally) secure.

In the computational setting the initial work of Yao [40] and Goldreich, Micali and Wigderson [17] has led to a large body of work [15, 16, 22]. In the past decade a number of practical schemes based on garbled circuits and oblivious transfer have emerged for the honest-but-curious adversarial model: Fairplay [7], Tasty [18] and VMCrypt [28]. But these schemes are for SMC and are inefficient and impractical when restricted to the special case of C-CBPS because of the need to handle Universal circuits [25]. Again, as mentioned before, the recent breakthrough in FHE [14] holds out hope for more efficient protocols for SMC but practical protocols are yet to be realized.

On the information-theoretic front the works of BenOr, Goldwasser and Wigderson [8] and Chaum, Crépeau and Dåmgard [10] proved completeness results for SMC. There have been additional improvements [11] using the subsequently developed notion of randomized encodings [2, 3]. The seminal work of Feige, Kilian and Naor [13], which lays the groundwork of this paper, is considered an interaction model. The FKN model (detailed in Definition 11, Subsection 2.3) is closely related to, but different from the C-CBPS model. For the FKN model [13] there are demonstrated protocols for predicates in NC^1 and also in NL. Though $\mathsf{NC}^1 \subseteq \mathsf{NL}$ [4, 29] the NC^1 protocol is the one we build on in this paper to achieve scalable and secure matching in the C-CBPS model.

The protocol of Feige, Kilian and Naor [13] in the FKN model is rendered impractical when used for the case of C-CBPS because of the need (for the broker) to compute universal circuits [33]. We explain this briefly: the FKN model involves the broker computing a known public function $f(\mathbf{P}, \mathbf{m})$ given the encrypted version of $\mathbf{m}$, data from the publisher, and the encrypted version of $\mathbf{P}$, data from the subscriber. C-CBPS is the special case where $f(\mathbf{P}, \mathbf{m}) = \mathbf{P}(\mathbf{m})$, i.e., $\mathbf{P}$ is a predicate (or the encoding of one) and f is a *universal* function that simulates $\mathbf{P}$ on $\mathbf{m}$. The need for universal circuits poses a barrier - both theoretical and practical. Though, some optimizations for universal circuits have been discovered [25], the current state of the art is that the subscriber is restricted to predicates of strictly sublogarithmic depth [31, 33] - more specifically, the predicate is constrained by the condition $d * (\log_2 s) = O(\log_2 n)$ where d is the depth and s the size of the (bounded fanin) subscriber predicate. In general, since s can be as large as 2^d this means that d is restricted to be $O(\sqrt{\log_2 n})$. In this paper we show how to bypass the need for a universal circuit and achieve $d = \Omega(\log_2 n)$ i.e. we can match any NC^1 predicate in the C-CBPS interaction model. We show how carefully constructed 2-decomposable randomized encodings [19–21] can be used to securely and efficiently simulate arbitrary circuits of logarithmic depth.

2 Preliminaries

2.1 Adversarial Model

There are 3 parties in our C-CBPS model – the broker, the publisher and the subscriber. The publisher and subscriber have a shared random string r known only to the two of them. Meta-data $\mathbf{m}$ is private to the publisher and predicate $\mathbf{P}$ is private to the subscriber. The publisher and subscriber each have a separate and private channel to the broker. They are to each send a single message to the broker from which the broker should be able to correctly, securely and efficiently compute the value of $\mathbf{P}(\mathbf{m})$ but learn absolutely nothing else. We work in the information-theoretic security model where we don't make any constraining assumptions on the computational resources available to the broker. However, the publisher and subscriber are restricted to be polynomial-time probabilistic Turing machines. In fact, given the shared randomness they are deterministic polynomial-time Turing machines. And in keeping with Kerckhoffs's principle [23] it is assumed that the broker knows the details of the algorithm/protocol being used, i.e., it knows everything except for $\mathbf{P}$, $\mathbf{m}$ and r. This adversarial model is captured in Figure 2.

2.2 Measure

We denote the length of the meta-data $\mathbf{m}$ as $n = |\mathbf{m}|$. In the case of the predicate $\mathbf{P}$ the relevant measure is the depth and we parameterize it as a multiple of $\log_2 n$, to

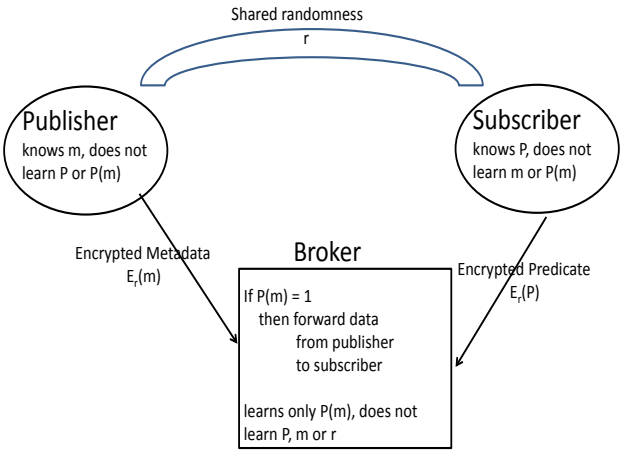

Fig. 2. Adversarial model

be precise we denote the depth of $\mathbf{P}$ by $\kappa \log_2 n$ where n is the parameter that asymptotically goes to infinity while κ is a constant. The reason for parameterizing in this way is that the obvious parameterization of the *size* of $\mathbf{P}$, in terms of the number of gates, is not relevant as it is still an open problem to handle arbitrary polynomial-sized predicate circuits in the information theoretic setting. We remind the reader that the defining contribution of this paper is showing how to handle circuits in NC^1 , i.e. logarithmic-depth circuits.

As we will see from the subsequent sections of this paper, the broker will get two (sub)sequences of group elements each from the publisher and subscriber that he will interlace and multiply together to obtain $\mathbf{P}(\mathbf{m})$. We denote by L the length of the sequence that the broker composes from the shares he receives. The efficiency question, thus, becomes given an n and a κ what is the smallest L that a given protocol achieves. In what follows we will see that, in the case of the protocol based on Valiant's universal circuit [33], $L = n^{\Omega(\kappa \log_2 n)}$ which is non-polynomial. In general the goal of this paper is to achieve L which will be a polynomial in n but the lower the degree of the polynomial the more efficient the protocol. We will show that UGP-Match achieves $L = n^{20\kappa+2} \log_2 n$. With FSGP-Match we bring this down to $L = 4n^{2\kappa+2}$ and then finally with OFSGP-Match we bring it down to $L = 2n^{2\kappa+1}$. We point out that these upper bounds are exact, i.e. we can analyze these constructions down to the exact constant and so do not need to employ the big-oh notation. For convenience we present our results in the form of a table.

Protocol	Complexity(L)	In words
Universal Circuit	$n^{\Omega(\kappa \log_2 n)}$	(super-poly-time)
UGP-Match	$O(n^{20\kappa+2} \log_2 n)$	(poly-time)
FSGP-Match	$4n^{2\kappa+2}$	"
OFSGP-Match	$2n^{2\kappa+1}$	"

Table 1. Complexities of the universal circuit approach and our 3 secure match protocols.

2.3 Terminology and Propositions

We set up some terminology and definitions; we also state some basic propositions that will be needed later. The notation for permutations is standard [38] while the notation for group programs follows [13]. We begin with a formal definition of the C-CBPS model.

Definition 1 (C-CBPS)**.** *The 3 parties in the C-CBPS model and their states of knowledge are captured in Figure 2. There is a single round of communication where the publisher and subscriber each send a private message ($E_r(\mathbf{m})$ and $E_r(\mathbf{P})$, respectively) to the broker who is then able to efficiently compute $\mathbf{P}(\mathbf{m})$ such that*
Correctness: *$\forall \mathbf{m}, \mathbf{P}$, and r, given the encrypted messages $E_r(\mathbf{m}), \mathbf{E_r}(\mathbf{P})$ the broker computes $\mathbf{P}(\mathbf{m})$ correctly all the time.*
Security: *$\forall \mathbf{m}, \mathbf{P}$, and r, given the encrypted messages $E_r(\mathbf{m})$, $E_r(\mathbf{P})$ the broker learns nothing whatsoever about $\mathbf{m}, \mathbf{P}$ (other than the value of $\mathbf{P}(\mathbf{m})$).*

We will develop some group-theoretic notation that is fairly standard. The notation is presented for the sake of completeness and is relevant when reading Appendix B.1 and Subsection 4.4. We use the language of multiplicative groups. For our purposes a *group* is a set of elements with a binary operation and inverses. We let G denote a generic multiplicative group. We will need G to be a non-solvable group (see Lemma 30 in Appendix B) and so we take it to be S_5 the symmetric group of permutations of a 5-element set, which is the smallest non-solvable group (see [39]). We will use the standard and implicit one-line notation (derived from Cauchy's two-line notation [38]) to represent a permutation. The permutation $\sigma \in S_5$ is represented as $(\sigma(1), \sigma(2), \ldots, \sigma(5))$ where $\sigma(i)$ stands for the position that $i \in S_5$ gets mapped to. σ can also be viewed as a function that maps i to $\sigma(i)$. The identity $1_{S_5} = (12345)$. We use the standard product of permutations defined as their composition as functions, i.e., $\sigma \cdot \pi$ is the function that maps i to $\sigma(\pi(i))$, (note that the rightmost permutation is applied to the argument first, in keeping with the con-

vention for function composition). Also, note that permutation groups S_k are not abelian for $k \geq 3$ and in particular S_5 is not abelian. A permutation $g \in S_5$ is said to be a *cycle* if its graph consists of exactly one cycle of length 5. For example, (53412) is a cycle because its graph is the cycle $1 \to 5 \to 2 \to 3 \to 4 \to 1$. When we use the product-of-cycles notation then we will explicitly have a subscript "cycle" at the end. For example $(24513) = (124)(35)_{\text{cycle}}$ represents the permutation $1 \to 2, 2 \to 4, 3 \to 5, 4 \to 1, 5 \to 3$ with graph $1 \to 2 \to 4 \to 1, 3 \to 5 \to 3$. As usual, g^{-1} denotes the *inverse* of the permutation $g \in S_5$. Given cycles $g = (g_1, g_2, \ldots, g_5)_{\text{cycle}}$ and $h = (h_1, h_2, \ldots, h_5)_{\text{cycle}}$, *rotator* $\rho(g, h)$ denotes the permutation $g_1 \to h_1, g_2 \to h_2, ..., g_5 \to h_5$ (see Lemma 26 in Appendix B). Given two permutations g, h $\gamma(g, h) = ghg^{-1}h^{-1}$ denotes their commutator. For the rest of this paper we will use the following fixed constants: $\alpha = (23451)$ and $\beta = (35421)$. The meticulous reader can verify that $\alpha^{-1} = (51234)$, $\beta^{-1} = (54132)$, $\gamma(\alpha, \beta) = (35214)$, $\gamma^{-1}(\alpha, \beta) = (43152)$ and $\rho(\alpha, \beta) = (13542)$, $\rho^{-1}(\alpha, \beta) = (15243)$.

Our protocols involve group programs, i.e., sequences of group elements and their products, motivating:

Definition 2 (Value of sequence)**.** *Given a sequence S of group elements g_1, g_2, ..., g_L, the* value *of the sequence $Value(S)$ is defined to be the product of the sequence elements in order, i.e.*

$$Value(S) = \prod_{i=1}^{L} g_i = g_1 g_2 \ldots g_L.$$

Definition 3 (Blinding)**.** *Given a sequence S of group elements g_1, g_2, ..., g_L, $\mathcal{BS}(S)$ is used to denote the distribution over sequences of the form*

$$g1 \cdot r_1, r_1^{-1} \cdot g_2 \cdot r_2, \ldots, r_L^{-1} \cdot g_L,$$

generated by choosing each $r_i, \forall i 1 \leq i \leq L-1$ uniformly and independently from G. We overload the term $\mathcal{BS}(S)$ to also refer to a specific sequence selected according to the distribution $\mathcal{BS}(S)$ and the context should be sufficient to resolve any ambiguity. The r_i are referred to as blinders *and* blinding *S refers to the act of selecting a sequence according to the distribution $\mathcal{BS}(S)$.*

Lemma 4 (Blinding Lemma)**.** *Given a sequence of group elements, S, of length L, the blinded sequence $\mathcal{BS}(S)$ has the following two properties:*
Preserves value*: $Value(\mathcal{BS}(S)) = Value(S)$*
Uniform distribution*: $\mathcal{BS}(S)$ is uniformly distributed over the space of all sequences of group elements of length L with the same value, i.e. for any sequence $S' = g'_1, g'_2 \ldots g'_L$ we have that*

$$Pr(\mathcal{BS}(S) = S' \,|\, Value(S') = Value(S)) = \frac{1}{|G|^{L-1}}$$

where the probability is measured over the random choices of the blinders.

The proof is presented in Appendix A.

The following definition, of a group program, is central to this paper and is presented in visual form in Figure 3.

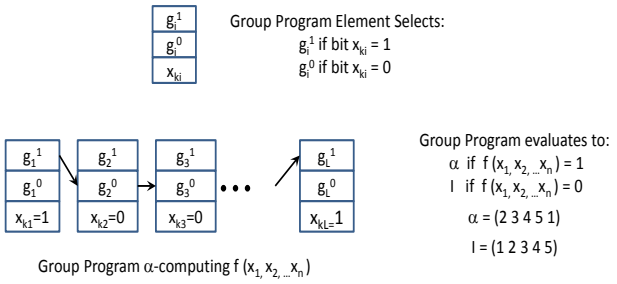

Fig. 3. The form of a Group Program

Definition 5 (Group Program)**.** *Let α be an element of S_5. A group program of length L is $(g_1^0, \ldots, g_L^0)$, $(g_1^1, \ldots, g_L^1)$, $(k_1, \ldots, k_L)$ where for any $i \in \{0, 1, \ldots, L\}$ and $j \in \{0, 1\}$: $g_i^j \in S_5$ and $k_i \in \{1, \ldots, n\}$. We say that this program α-computes $f : \{0,1\}^n \to \{0,1\}$ if $\forall x$,*

$$f(x) = 1 \Rightarrow \prod_{i=1}^{\ell} g_i^{x_{k_i}} = \alpha$$

$$f(x) = 0 \Rightarrow \prod_{i=1}^{\ell} g_i^{x_{k_i}} = 1_{S_5};$$

which we write compactly as $\forall x : \prod_{i=1}^{\ell} g_i^{x_{k_i}} = \alpha^{f(x)}$.

We will consider predicates represented as n-input, single output (bounded fan-in) circuits of AND($\wedge$ two-input), OR($\vee$ two-input) and NOT($\neg$ single input) gates. Our definition of the depth of a circuit is slightly non-standard in that we ignore NOT($\neg$) gates. This is because of the Barrington Transform (to be elaborated below) which transforms a circuit into a group program whose length depends only on the depth of the circuit in terms of AND($\wedge$) and OR($\vee$) gates.

Definition 6 (Depth of a Circuit)**.** *The depth of a circuit is defined to be the number of AND and OR gates in the longest path from an input to the output. (NOT gates do not count towards depth).*

In his seminal paper [6], on the way to showing that NC^1 is computable by fixed-width branching programs, Barrington showed that any logarithmic-depth circuit can be transformed into a polynomial-length group program [34, 36] - a transformation we term the Barrington Transform:

Theorem 7 (Barrington Transform)**.** *Any circuit of depth d can be transformed into a group program of length 4^d that α-computes the same function as the circuit.*

For the sake of completeness we present the proof in Appendix B.1. The proof, presented as a series of lemmas, details the Barrington Transform showing how to transform a circuit into a group program [34, 35].

Corollary 8. *The Barrington Transform transforms any n-input single output circuit of depth $C \log_2 n$ into a group program of length n^{2C}.*

The above corollary follows directly from Theorem 7.

We use an extension of randomized encodings [1, 3, 19] (that played a significant role in the breakthrough showing the feasibility of cryptography in NC^0 [2]):

Definition 9 (Randomized encoding)**.** *Function $f(x)$ has a randomized encoding $\hat{f}(x,r)$, where r is a random string, if there exist two efficiently computable (deterministic, polynomial-time) algorithms REC and SIM such that*
Correctness: *$\forall x, r$, given $\hat{f}(x,r)$ REC recovers $f(x)$, i.e. $REC(\hat{f}(x,r)) = f(x)$.*
Security: *$\forall x$, given $f(x)$ and r' (random coins), SIM produces a distribution identical to $\hat{f}(x,r)$, i.e., the distribution of $SIM(f(x), r')$ is identical to the distribution of $\hat{f}(x,r)$.*

Randomized encodings generalize naturally, in the context of 2-party secure computation, to *2-decomposable randomized encodings* or 2-DREs, a definition implicit in prior works including [13]. A stronger definition of decomposable randomized encodings has appeared before [20, 21]. We believe that the following is the first explicit definition:

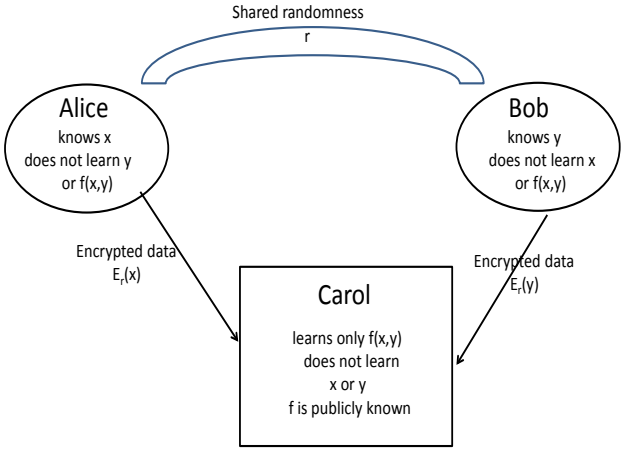

Fig. 4. FKN model

Definition 10 (2-DRE)**.** *A function $f(x,y)$ is said to have a 2-decomposable randomized encoding (2-DRE) $\langle \hat{f}_1(x,r), \hat{f}_2(y,r) \rangle$, where r is a (*shared*) random string, if there exist two efficiently computable (i.e. deterministic and polynomial-time) algorithms REC and SIM such that*
Correctness: *$\forall x, y, r$, given $\langle \hat{f}_1(x,r), \hat{f}_2(y,r) \rangle$ REC recovers $f(x,y)$, i.e., $REC(\langle \hat{f}_1(x,r), \hat{f}_2(y,r) \rangle) = f(x,y)$.*
Security: *$\forall x, y$, given $f(x,y)$ and r' (random coins), SIM produces a distribution identical to $\langle \hat{f}_1(x,r), \hat{f}_2(y,r) \rangle$, i.e., the distribution of $SIM(f(x,y), r')$ is identical to the distribution of $\langle \hat{f}_1(x,r), \hat{f}_2(y,r) \rangle$.*

We rely crucially on the model and construction presented in [13]. We define the FKN model and show how protocols for it are essentially equivalent to 2-DREs:

Definition 11 (FKN model)**.** *The three parties in the FKN model and their states of knowledge are captured in Figure 4. There is a single round of communication where Alice and Bob each send a private message ($E_r(x)$ and $E_r(y)$, respectively) to Carol who is then able to efficiently compute the publicly known function $f(x,y)$ such that*
Correctness: *$\forall x, y$, and r, given the encrypted messages $E_r(x), E_r(y)$ Carol computes $f(x,y)$ correctly all the time.*
Security: *$\forall x, y, r$, given encrypted messages $E_r(x), E_r(y)$ Carol learns nothing (whatsoever about x, y) other than $f(x,y)$.*

The phrase "learns nothing" in the definition above is from [13] and denotes that the protocol is per-

fect zero-knowledge, i.e. given random coins and $f(x,y)$, Carol can simulate perfectly the distribution $< E_r(x), E_r(y) >$ in deterministic polynomial-time.

Lemma 12 (Equivalence of FKN and 2-DRE)**.** *There is a 1-1 isomorphism between protocols for the FKN model and 2-DREs.*

Proof. We will first see that a 2-DRE of the function $f(x,y)$ gives rise to a protocol for the FKN model. Let Alice compute and send $\hat{f}_1(x,r)$ to Carol while Bob computes and sends $\hat{f}_2(y,r)$ to Carol. From the Correctness property it follows that Carol can run $\mathrm{REC}(\langle \hat{f}_1(x,r), \hat{f}_2(y,r)\rangle)$ to compute $f(x,y)$ correctly all the time. And from the Security property we get that Carol learns nothing but the value $f(x,y)$.

Similarly, given a protocol for the FKN model that computes the publicly known function $f(x,y)$ it is easy to see that the Correctness and Security properties carry over by setting $E_r(x) = \hat{f}_1(x,r)$ and $E_r(y) = \hat{f}_2(x,r)$. □

We present the definition of a *universal function* below.

Definition 13 (Universal Function)**.** *A universal function $\mathcal{UF}$ has two inputs x and y where y is the encoding of a function for which x is a suitable input. The output of $\mathcal{UF}(x,y)$ is defined to be $y(x)$.*

Similarly, *universal circuits* are circuits that simulate circuits given their encodings:

Definition 14 (Universal Circuit)**.** *A universal circuit $\mathcal{UC}$ has two inputs x and y where y is the encoding of a circuit for which x is a suitable input. The output of $\mathcal{UC}(x,y)$ is defined to be $y(x)$, in other words the universal circuit outputs the result obtained from simulating circuit y on input x. (Universal circuits are the circuit equivalent of universal Turing Machines).*

Our central idea is to bypass universal circuits so we will not be utilizing this definition except to explain (in Section 3) how we gain efficiency by avoiding them.

One can naturally apply the notion of a 2-DRE to a universal function ($\mathcal{UF}$) $f(x,y) = y(x)$.

And analogous to the correspondence expressed in Lemma 12 we have the following correspondence.

Lemma 15 (C-CBPS ≡ 2-DRE of $\mathcal{UF}$)**.** *There is a 1-1 isomorphism between protocols for the C-CBPS model and 2-DREs of Universal Functions.*

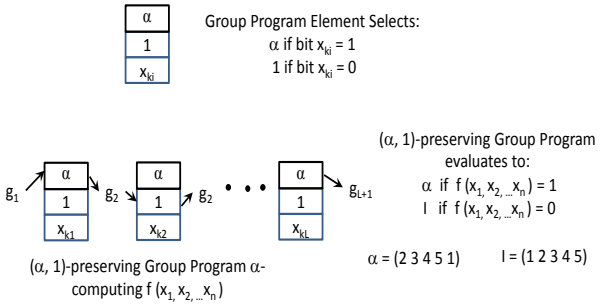

Fig. 5. $(\alpha, 1)$-preserving Group Program

Proof. The proof is very similar to that of Lemma 12 and so we provide only the key aspects of the correspondence as the details are straightforward. x, y correspond to $\mathbf{m}, \mathbf{P}$ respectively. $\hat{f}_1(x,r), \hat{f}_2(y,r)$ correspond to $E_r(\mathbf{m}), \mathbf{E_r}(\mathbf{P})$ respectively. And the Correctness and Security properties carry over in a direct way. □

We now present some definitions that clarify the "fixed structure" construction alluded to earlier, and how it relates to 2-DRE of universal functions. The following definition of a $(\alpha, 1)$-preserving group program is presented in visual form in Figure 5.

Definition 16 ($(\alpha,1)$-preserving group programs)**.** *An $(\alpha,1)$-preserving group program of length L is $(g_1, g_2, \ldots, g_{L+1})$, $(k_1, k_2, \ldots, k_L)$ where for any i: $g_i \in S_5$ and $k_i \in \{1, \ldots, n\}$. We say that this $(\alpha,1)$-preserving group program α-computes $f : \{0,1\}^n \to \{0,1\}$ if $\forall x$,*

$$f(x) = 1 \Rightarrow (\prod_{i=1}^{\ell} g_i \cdot \alpha^{x_{k_i}}) g_{L+1} = \alpha$$

$$f(x) = 0 \Rightarrow (\prod_{i=1}^{\ell} g_i \cdot \alpha^{x_{k_i}}) g_{L+1} = 1_{S_5};$$

which we can write compactly as $\forall x : (\prod_{i=1}^{\ell} g_i \cdot \alpha^{x_{k_i}}) g_{L+1} = \alpha^{f(x)}$.

Definition 17. (Index sequence of $(\alpha,1)$-preserving group program) *Given $(g_1, g_2, \ldots, g_{L+1})$, $(k_1, k_2, \ldots, k_L)$, an $(\alpha,1)$-preserving group program of length L, its index sequence is $(k_1, k_2, \ldots, k_L)$.*

Analogous to Theorem 7 we have

Theorem 18. ($(\alpha,1)$-preserving Barrington Transform) *Any circuit of depth d can be transformed into*

a $(\alpha, 1)$-preserving group program of length 4^d that α-computes the same function as the circuit.

For the sake of completeness we present the proof in Appendix B.2.

The definition of a fixed structure group program given below is crucial to improving the efficiency of the ideas presented in this paper and making them practical.

Definition 19 (Fixed structure group program)**.** *If a class of functions (say, the class of n-input single output functions computable by circuits of depth $\kappa \log_2 n$) can all be transformed into $(\alpha, 1)$-preserving group programs with the exact same index sequence (and hence the exact same length) then the resulting class of group programs is said to have a fixed structure. In a slight abuse of language we refer to the class itself as a fixed structure group program.*

3 Overview

Before explaining the bottleneck that this paper has overcome we first briefly sketch the protocol for the FKN model presented in [13] for publicly known functions $f(x, y)$ computable as logarithmic-depth circuits. The group program equivalent of f is computed (using Barrington's Transform [6]) by both Alice and Bob from the corresponding circuit. Each of them instantiates their share of the group program based on their respective input (x for Alice and y for Bob). Then each of them blinds their share in coordinated fashion using the shared randomness. Finally, each of Alice and Bob sends their respective blinded shares to Carol who puts them together to form the final blinded sequence whose value she computes to obtain $f(x, y)$. See Figure 6.

What [13] essentially demonstrate is the construction of a 2-DRE $\hat{f}$ of f, as elaborated in Lemma 12. Of course, the notion of 2-decomposable randomized encodings arose much later in the work of Ishai and Kushilevitz [19–21], but it gives us a convenient language to think about such protocols. The individual shares, constructed and, sent to Carol by Alice and Bob are just the two parts of the 2-DRE, namely $\hat{f}_1(x, r)$ and $\hat{f}_2(y, r)$. REC guarantees that Carol is able to learn $f(x, y)$ while SIM guarantees that she learns nothing beyond that.

Now, we explain the *universal circuits* bottleneck that we claim to have overcome. Recall that in our (C-

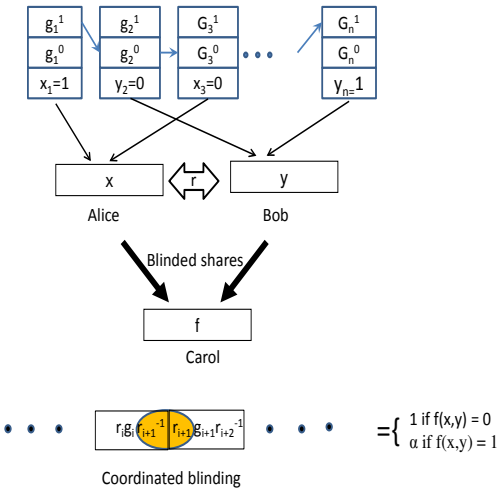

Fig. 6. Protocol for FKN model, see [13].

CBPS) setting f is not just any function but it is a *universal function* where $f(x, y) = y(x)$. The term *universal* comes from the fact that f is effectively simulating y on x and so the natural question arises - how efficiently can this simulation be done? In other words, given that we are restricted to have only polynomial-size group programs, what restriction does this put on the class of functions represented by y? The best known construction of universal circuits is due to Valiant [33] who shows that if y were a circuit of size s and depth d then it must satisfy $d \log s = O(\log n)$ (for the resulting Barrington Transform to produce a polynomial-length group program). Observe that this constraint on d and s automatically prevents y from representing circuits in NC^1 because $s \geq d$ and hence d is forced to be $o(\log n)$. Sanders, Young and Yung [31] who utilize Valiant's universal circuits construction, mention that y could be the class of functions represented by circuits of depth $\sqrt{\log n}$ and size $2^{\sqrt{\log n}}$ - note that this is believed to be a strict subset of the class of predicates in NC^1 .

Our main contribution is in showing how we can bypass the universal circuits bottleneck and instead use group programs to improve the efficiency of simulation. We present two main constructions. The first, UGP-Match, which is primarily of theoretical interest, shows how we can handle all of NC^1 by encoding the subscriber's predicate as a group program (using the Barrington Transform) and constructing a universal group program using the Barrington Transform (again). This construction has a conceptually clean proof but it is impractical since the double invocation of the Barrington Transform results in a final group program whose length

is a very high-degree polynomial in n. Our second construction, FSGP-Match, which is potentially practical, uses a fixed structure construction to avoid one invocation of the Barrington Transform. Rather than construct a universal group program using the Barrington Transform we directly construct a fixed structure group program. By avoiding the inefficiencies arising from use of the Barrington Transform we obtain a final group program whose length is a relatively low-degree polynomial in n. In this second construction of FSGP-Match we need a special group program - the *selector group program* - which we construct by applying the Barrington Transform to a *selector circuit*. Lastly, we present our most efficient construction OFSGP-Match by directly building the selector group program and bypassing the use of the Barrington Transform yet again. We also describe our Java-based implementation and provide performance results that point to the potential of our techniques for real-world applications. Unlike our proofs (which are in the information-theoretic setting) the implementation reuses randomness providing guarantees dependent on the cryptographic strength of the pseudo-random generator, in the computational model.

4 Protocols and Proofs

4.1 UGP-Match

As explained in Section 3 the main idea in the UGP-Match construction is to use the Barrington Transform to encode the predicate $\mathbf{P}$ (which is representable as a circuit in NC^1) as a polynomial-length group program that indexes into the metadata bits $\mathbf{m}$. The (publicly known) circuit f in the protocol for the FKN model [13] (see Figure 6) is chosen to be such that at the lowest level it first selects the appropriate group program element based on the index and value of the corresponding bit in $\mathbf{m}$ and having selected all the group program elements it then multiplies them together using a standard divide and conquer or balanced binary tree based approach, see Figure 7. In a nutshell, UGP-Match uses the protocol from [13] for the FKN model, with f being a circuit that takes as input the metadata $\mathbf{m}$ and the predicate $\mathbf{P}$ encoded as a group program and outputs the result of simulating $\mathbf{P}$ on $\mathbf{m}$. We present UGP-Match more formally below. We refer to the circuit representing f as the UGP-Match Circuit. We refer to the final group program that the broker assembles as the Universal Group Program because it is essentially a Group Program that simulates the group program representing $\mathbf{P}$.

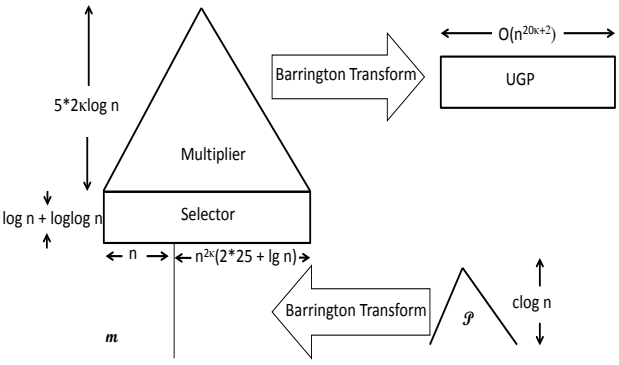

Fig. 7. UGP-Match Circuit

UGP-Match

1. The publisher and subscriber register with the broker the precise form of their inputs. In particular the publisher must specify the number n of bits of $\mathbf{m}$ the metadata (if there are fewer relevant bits, the remaining bits can be padded with dummy bits). And the subscriber specifies $L = n^{2\kappa}$ the length of the group program (in terms of number of group element pairs (not bits) along with the bit of $\mathbf{m}$ that each group element pair is dependent on). Everybody is coordinated on the choice of the non-solvable group - S_5 - in which to carry out their computations. In order to optimize the group program size we encode the group elements in unary using $5 \times 5 = 25$ bits.
2. The broker now computes the UGP-Match Circuit f (essentially a Select block followed by a divide-and-conquer Multiply block, see Figure 7). It then applies the Barrington Transform to create the corresponding group program. Each group element pair in this group program is dependent either on a subscriber bit or on a publisher bit. The broker hands back the entire group program to both the publisher and the subscriber.
3. The publisher and subscriber know which pairs belong to each. They have already coordinated their pre-shared randomness. Depending on the value of their individual bit they pick the corresponding element of the pair and then blind it appropriately.

They then give their respected blinded elements to the broker.

4. The broker puts all the blinded elements together in the right sequence and multiplies them. If he gets α he forwards along the (encrypted) data from the publisher to the subscriber, else he withholds it.

Theorem 20. *Given metadata of size at most n and predicate of depth at most $\kappa \lg n$, UGP-Match is an information-theoretically secure protocol for the C-CBPS model with complexity $O(n^{20\kappa+2} \log_2 n)$.*

Proof. The proof that UGP-Match is correct and secure follows directly from the proof in [13] for the FKN model.

All that remains to do is to bound the complexity of UGP-Match. We now provide a detailed description of the UGP-Match Circuit f. See Figure 7. This circuit takes in as input $\mathbf{m}$ and the bit-representation of the group program obtained from applying the Barrington Transform (see Theorem 7) to $\mathbf{P}$. The group program is a sequence of $L = n^{2\kappa}$ group program elements each of which is two group elements and an index (into $\mathbf{m}$. We represent the group elements which are permutations of S_5 in unary, e.g., we would represent the permutation (23451) as the bit sequence 00010 00100 01000 10000 00001 (the spacings are placed for convenience of reading). We note that this is not the most economical representation in terms of bit-length but what we are ultimately looking to minimize is the length of the final group program, i.e. the depth of the UGP-Match Circuit and for that purpose this representation is close to optimal. Each of the indices in binary is represented using $\lg n$ bits. The UGP-Match Circuit is a Select block followed by a divide-and-conquer Multiply block. In the Select block, corresponding to each group program index the value of the bit of $\mathbf{m}$ is extracted using a mux (multiplexer, see [37]) which is a circuit of depth $\lg n + \lg \lg n$. The corresponding group program element is then selected using a circuit of depth 2. Multiplying two group program elements takes a circuit of depth 5 and hence the Multiply block has depth $(5 \times 2\kappa) \lg n$ (recall that the group program $\mathbf{P}$ has length $n^{2\kappa}$). Thus the total depth of the UGP-Match Circuit is $(10\kappa + 1) \lg n + \lg \lg n + 2$. The Barrington Transform (see Theorem 7) of UGP-Match Circuit gives us a final group program under the FKN model of length $\tilde{O}(n^{20\kappa+2})$. □

Corollary 21. *UGP-Match is an efficient and secure protocol for matching any predicate in* NC^1 .

4.2 Fixed Structure Group Programs (FSGP)

We now state and prove the key lemma concerning fixed structure group programs.

Lemma 22 (FSGP yield 2-DRE of UF)**.** *Fixed structure group programs are convertible to 2-DREs of universal functions with no loss of efficiency.*

Proof. The conversion of a fixed structure program to a 2-DRE of a universal function is fairly straightforward involving instantiation and coordinated blinding.

Consider a class of predicates $\mathbf{P}$ convertible to fixed structure group programs. Let $(g_1, g_2, \ldots, g_{L+1})$, $(k_1, k_2, \ldots, k_L)$ be the fixed structure group program, i.e., a $(\alpha, 1)$-preserving group program with a fixed index sequence (the index sequence is the same independent of the specific function that the group program is computing though the interstitial group elements $(g_1, g_2, \ldots, g_{L+1})$ would depend on the specific function). We now need to demonstrate two functions $\hat{f}_1(\mathbf{m}, \mathbf{r})$ and $\hat{f}_2(\mathbf{P}, \mathbf{r})$ satisfying the requirements of Definition 10, with the additional constraint of universality, namely that $f(\mathbf{m}, \mathbf{P}) = \mathbf{P}(\mathbf{m})$.

Given a specific instance of metadata $\mathbf{m}$ the function $\hat{f}_1$ first selects α or 1 as appropriate for each of the index sequence pairs; note that this is done independent of the specific predicate $\mathbf{P}$. Then $\hat{f}_1$ blinds the resulting sequence using the randomness r. Similarly, depending on the specific predicate $\mathbf{P}$ $\hat{f}_2$ converts to the fixed structure group program and gets a specific instantiation of the interstitial group elements $(g_1, g_2, \ldots, g_{L+1})$. Then this sequence is blinded by $\hat{f}_2$ using r. Observe that the fixed structure is crucial for the construction of $\hat{f}_1$ so that it is independent of $\mathbf{P}$.

It remains to prove Correctness and Security as per Definition 10. First, the existence of REC and Correctness follows because of the appropriate cancelation of the blinders and so $\mathbf{P}(\mathbf{m})$ can be recovered exactly from multiplying the elements of the final assembled group program. Next, the existence of SIM and Security - observe that independent of the specific instance the broker receives shares of the same form from both publisher and subscriber and by Lemma 4 (Blinding Lemma) these shares are uniformly distributed over all possible sequences with same final value and therefore SIM can generate the outputs with the same distribution as $\hat{f}_1$ and $\hat{f}_2$ by generating all but one element of the sequence completely and uniformly at random, then generating the last element so that the value of the se-

quence is the given final value, and then splitting the group program into the two parts corresponding to $\hat{f}_1$ and $\hat{f}_2$. □

Observe that an implication of the above Lemma is that the fixed structure of the group program implicitly encodes a universal function.

The below corollary follows from the two lemmas, Lemma 22 and 15.

Corollary 23. *Fixed structure programs yield secure C-CBPS protocols with no loss of efficiency.*

4.3 FSGP-Match

With Lemma 22 and Corollary 23 in hand, all we need to do is to demonstrate a fixed structure group program of the right complexity. We give an intuitive description of the main idea. Applying the $(\alpha, 1)$-preserving Barrington Transform to the predicate $\mathbf{P}$ yields an $(\alpha, 1)$-preserving group program but not a fixed structure group program because the index sequence would vary depending on the predicate $\mathbf{P}$. So, then we substitute each instance of a group program pair in this group program by a fixed $(\alpha, 1)$-preserving group program block we call the Fixed Selector Group Program. This creates a fixed structure. And by padding appropriately we can fix the length to be independent of the inputs as well. It remains to describe the Fixed Selector Group Program. The Fixed Selector Group Program is obtained by applying an $(\alpha, 1)$-preserving Barrington Transform to the Fixed Selector Circuit. Figure 8 is a visual description of the Fixed Selector Circuit. The Fixed Selector Circuit is an OR of ANDs. It has one AND gate for each bit of the metadata $\mathbf{m}$ as input. The other input to the AND gate is a "new" bit under the control of the subscriber. Since the subscriber knows the predicate $\mathbf{P}$ it knows exactly which bit of the input ($\mathbf{m}$) it needs in that particular block (based on the index of the group program pair that the block is replacing). Once the subscriber instantiates these bits approrpriately and blinds then the Fixed Selector Group Program is just a block with a fixed index sequence since all the dependence on the predicate $\mathbf{P}$ is factored into the interstitial group elements. Thus the overall index sequence is fixed. Note that for blocks in the padding the subscriber can set all new bits to 0 thus effectively reducing the padding part of the group program to a no-op, something that evaluates to 1.

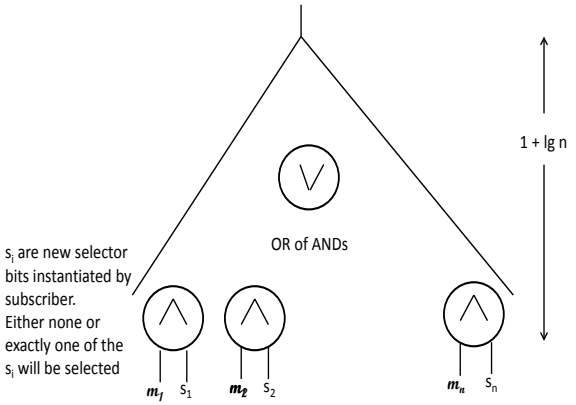

Fig. 8. Fixed Selector Circuit

FSGP-Match

1. The publisher and subscriber register with the broker the precise form of their inputs. In particular the publisher must specify the number n of bits of $\mathbf{m}$ the metadata (if there are fewer relevant bits, the remaining bits can be padded with dummy bits). And the subscriber specifies $D = 2\kappa \lg n$ the maximal depth of the predicate $\mathbf{P}$.
2. The broker now computes the form of the fixed structure group program by applying the $(\alpha, 1)$-preserving Barrington Transform to any circuit of depth D to create the corresponding $(\alpha, 1)$-preserving group program. Again, only the form of the group program is relevant at this stage, not the values of the specific interstitial group elements or indices in the index sequence. It then replaces each group program pair with the Fixed Selector Group Program and returns the entire composite group program back to the publisher and subscriber. At this point the values of the interstitial group elements do not matter but the index sequence, which is fixed, does matter. The program is an $(\alpha, 1)$-preserving group program with indices that index into the metadata bits.
3. The publisher selects one of α or 1 depending on the value of the corresponding metadata bit for each group program pair, blinds them and sends back to the broker. The subscriber, applies the $(\alpha, 1)$-preserving Barrington Transform to the specific instance $\mathbf{P}$ to get the concrete values for the interstitial group elements, appropriately sets the "new" bits for the Fixed Selector Group Program blocks, and blinds the entire sequence and sends to the broker.

4. The broker puts all the blinded elements together in the right sequence and multiplies them. If he gets α he forwards along the (encrypted) data from the publisher to the subscriber, else he withholds it.

Theorem 24. *Given metadata of size at most n and predicate of depth at most $\kappa \lg n$, FSGP-Match is an information-theoretically secure protocol for the C-CBPS model with complexity $4n^{2\kappa+2}$.*

Proof. FSGP-Match is obtained by applying the $(\alpha, 1)$-preserving Barrington Transform to the predicate $\mathbf{P}$ to yield an $(\alpha, 1)$-preserving group program where each instance of a group program pair is substituted by the Fixed Selector Group Program. It is clear that FSGP-Match produces a fixed structure group program and hence it follows from Lemma 22 that it is correct and secure.

All that remains to do is to bound the complexity of FSGP-Match. Recall that group program pair in the group program (of length $n^{2\kappa}$ is substituted by the Fixed Selector Group Program. The Fixed Selector Group circuit (see Figure 8) has depth $1 + \lg n$ and hence the resulting Fixed Selector Group Program has length $4n^2$ which means that the final group program has a length $4n^2 \times n^{2\kappa} = 4n^{2\kappa+2}$. □

4.4 OFSGP-Match

Rather than constructing the fixed structure by taking the Barrington Transform of the Fixed Selector Circuit we directly construct an optimized Fixed Selector Group Program. See Figure 9.

Theorem 25. *Given metadata of size at most n and predicate of depth at most $\kappa \lg n$, OFSGP-Match is an information-theoretically secure protocol for the C-CBPS model with complexity $2n^{2\kappa+1}$.*

Proof. The proof is similar to the proof of Theorem 24. Instead of converting a Fixed Selector Circuit to a group program we directly construct an optimized $(\alpha, 1)$-preserving Fixed Selector Group Program. Corresponding to the i-th AND gate of the Fixed Selector Circuit we create a subsequence of the group program consisting of two copies of group elements (α or 1_{S_5}) indexed by the i-th bit of the metadata $\mathbf{m}$, surrounded by (3) interstitial elements on either side.

For the sake of completeness we show explicitly that the group program subsequences when concatenated will compute to the value corresponding exactly to the one selected bit.

In the case when the bit x_i is not selected then it is easy to see that the resulting subsequence $1_{S_5} \cdot \alpha^{x_i} \cdot \rho^{-1}(\alpha^{-1}, \alpha) \cdot \alpha^{x_i} \cdot \rho(\alpha^{-1}, \alpha) = 1_{S_5}$ independent of the value of whether bit x_i is 0 or 1.

And, when x_i is selected then by the choice of the rotator (see Lemma 26) we have that the resulting subsequence $\rho^{-1}(\alpha^2, \alpha) \cdot \alpha^{2x_i} \cdot \rho(\alpha^2, \alpha) = \alpha^{x_i}$.

The Optimized Fixed Selector Group Program (see Fig. 9) has length $2n$ which means that the final group program has a length $2n \times n^{2\kappa} = 2n^{2\kappa+1}$. □

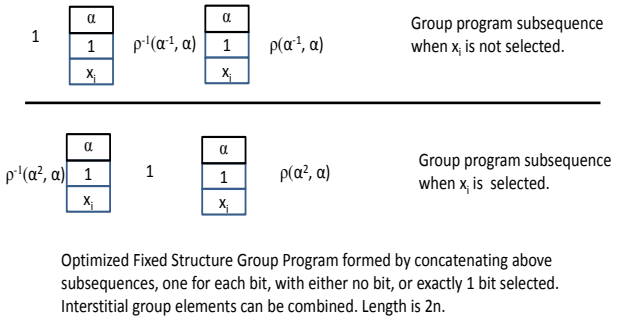

Fig. 9. Optimized Fixed Selector Group Program

5 Implementation

We have made available two open-source implementations of OFSGP-Match. Our first prototype in Scheme language, based on the jScheme interpreter for integration with Java programs, allowed us to explore various design alternatives. For example, we explored different representations for S_5, and alternative constructions of the UGP-Match to simulate Group Programs. We ported the OSFGP-Match portion to work with JavaScript for delivery over the Web [26]; we make use of the BiwaScheme interpreter and a random number generator from the Stanford Javascript Cryptographic Library. The Scheme code is short (∼115 lines excluding the pre-computed tables), and serves as a pedagogic illustration.

We have released a faster and extensible implementation of OFSGP-Match in Java [27], which is suitable for integration with enterprise applications, e.g., confidential inter-agency information sharing. We describe this implementation next.

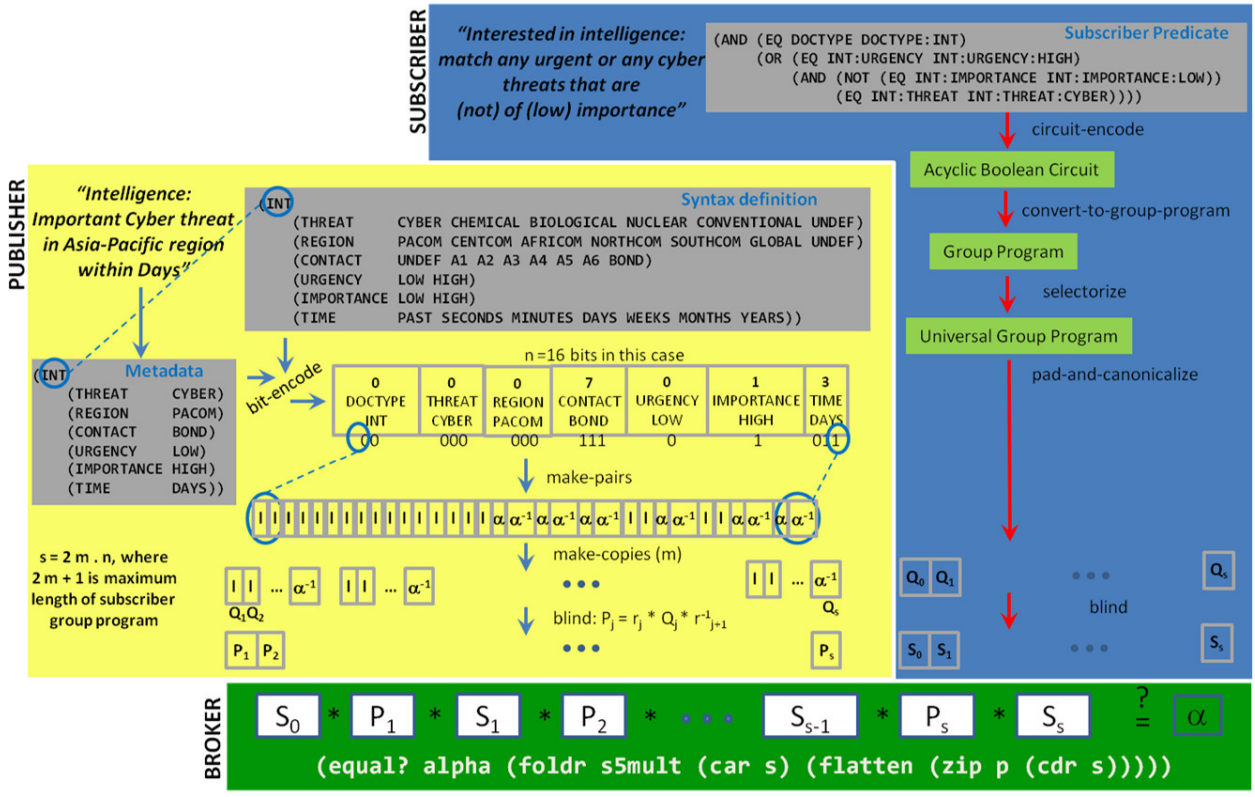

Fig. 10. Confidential Publish/Subscribe Example using OFSGP-Match

Operations in S_5: We use a byte to represent each of the 120 elements in S_5; although 7 bits would suffice, byte-based operation is more convenient. Our mapping uses 0 to encode *identity* in the group, which is convenient when preparing a sequence of bytes (e.g., for padding), since Java clears byte arrays upon allocation. For speed, we have implemented the inverse and multiplication operations as table look-ups. We generate random sequences in S_5 for FKN blinding.

Group Program Construction: We apply Barrington's transform to generate Group Programs in S_5. We implement a number of modular primitives including *inversion* (NOT), *conjunction* (AND), *concatenation*, *pin* (input bit), *constant*, and *BitOpBlock* (a block of bit-wise operations to operate on bit vectors). We have selected low-depth constructions of circuit building-blocks including IDENTITY, NOT, AND, OR, NAND, NOR, XOR, SELECT (IF-THEN-ELSE), ADD, SUBTRACT, EQUAL, SHIFT-LEFT, SHIFT-RIGHT, LESSER, LESSER-OR-EQUAL, GREATER, and GREATER-OR-EQUAL, and we generate the corresponding Group Programs. These functions are called by the expression parser, which we describe below. Our Java implementation generates the Group Program directly from the Boolean expression, with no explicit step (seen in the Scheme prototype) that outputs intermediate circuits.

Representation/Parsing of Metadata and Interests: For extensibility, we make use of XML Schema and instances to represent metadata from publishers and interests from subscribers. We use the Java Architecture for XML Binding (JAXB) for mapping Java classes to XML representations. JAXB is useful when the specification is complex and changing, and manual synchronization of XML Schema with Java classes is time consuming and error-prone. JAXB's `xjc` tool generates Java code—to marshal objects into XML and to unmarshal XML into objects—directly from XML Schema.

Each metadata instance must include a `RECORDTYPE` field. Currently, we support the following types of fields in the metadata: enumerations, fixed-length strings, and signed integer types including bytes, short, int, and long. Support for other types such as variable length strings, floating-point numbers, and more complex data types is the subject of future work. For each metadata instance (in XML), the parser generates a bit vector; it determines the size and structure of the bit vector from

the XML Schema. It computes the field sizes (in bits) required for each enumerated type, integer type, and fied-length string type; it also generates indices for enumerated field values in the metadata instances.

Each subscriber interest expression is tree-structured with a root `EXPR` element, which includes an operator field and child `EXPR` elements recursively; leaf `EXPR` elements specify either a field name or a field code in the operator field, and include a type field and a value field. The parser processes the expression by invoking Group Program construction functions corresponding to the operator field of each non-leaf element, and creating Group Programs corresponding to bit-vectors for enumeration codes or constants specified in leaf elements.

We include examples of metadata and interests for two domains – intelligence reports and stock tickers. Developers can specify XML Schema for other domains as needed for their particular applications, and generate parsing code using `xjc`.

OFSGP-Match: We have implemented functions for the steps of the OFSGP-Match protocols performed by the publisher, subscriber and broker.

The publisher parses the metadata instance to generate a bit vector of n bits, and produces a pair of group elements from S_5 for each bit for use with the selector. Then it replicates the sequence of $2n$ bytes $n^{2\kappa}/2$ times. It then blinds the total sequence using the FKN protocol to send to the broker for a secure match. A separate blinded sequence is prepared for each subscription.

The subscriber parses the interest expression to generate a Group Program. References to metadata bits (from the publisher) are replaced with selector blocks, and the resulting Group Program is canonicalized to a form where alternating elements are from publisher and subscriber by multiplying subscriber elements together as applicable. The sequence is padded to a fixed maximum length. The resulting sequence is blinded using the FKN protocol to send to the broker, once for each match.

The broker interleaves the sequences from the publisher and subscriber and then multiplies them to evaluate a match. It is a match if the result equals α, a given 5-cycle in S_5, and the content item is passed along to the subscriber. If the result is identity in S_5, then it is not a match. Any other result is an error condition.

End-to-end Publish/Subscribe Example: An end-to-end example of confidential publish/subscribe using OFSGP-Match is illustrated in Figure 10 showing steps at a publisher, a subscriber, and a broker. In this example, the metadata from the publisher is an intelligence report anticipating *an important cyber threat in the Asia-Pacific region within days*; the subscription is an interest in *intelligence reports that match any urgent events or cyber threats that are not of low importance*. The software release includes XML files for the interest and metadata, and a test to invoke and time the publisher, subscriber and broker functionality. For randomness, we use SHA1PRNG from the Java security library.

Our implementation of OFSGP-Match is reusable. A complete application system using this OFSGP-Match implementation must include functionality for: (i) securely exchanging seeds for the PRNG between each publisher and each subscriber for each subscription, (ii) preparing blinded instances of the publisher portion of the OFSGP from the metadata, one instance for each subscription, (iii) preparing multiple blinded instances of the subscriber portion of the OFSGP from the interest expression, one instance for each match, (iv) protocol fields that allow the broker to know whether the blinded OFSGP sequences from a publisher and a subscriber have been prepared with the same shared secret, and can hence be evaluated, (v) low-watermark thresholds at the broker and signaling to the subscriber to ensure that adequate prepared interests are available for matching with arriving publications, (vi) integrity checks for the protocol data units, and (vii) client-server network protocols.

6 Performance

We present measurements for the example presented in Figure 10. In this case, a metadata length of $n = 32$ bits and a circuit depth of $d = 5$ (i.e., $\kappa = 1$) suffice. The broker performs $2n^{2\kappa+1} = 65536$ group multiplications per match. The publisher produces a blinded sequence of length 32768 bytes per subscription and the subscriber produces a blinded sequence of length 32769 bytes for each metadata item to be matched. The time required for each step in OFSGP-Match for a typical run is shown in Table 2. The system used for the test is a laptop with a Intel Core i3-4005U processor (2 cores with 2 threads each) runing at 1.7GHz, and 6GB of memory. The operating system is Microsoft Windows 8.1 and the Java platform is Oracle Java SDK version 1.8.

We characterize the anticipated performance of a complete system numerically using a parametric model. In addition to the parameters, n, d, and κ, we define W_p, the network throughput between the publisher and

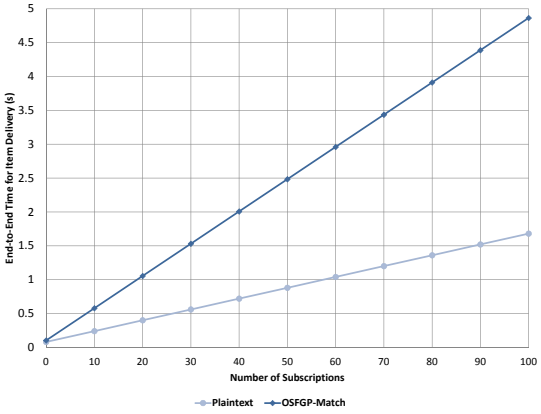

Fig. 11. End-to-end time from new content arrival to delivery of content to all matching subscriptions: plaintext vs. OFSGP-Match

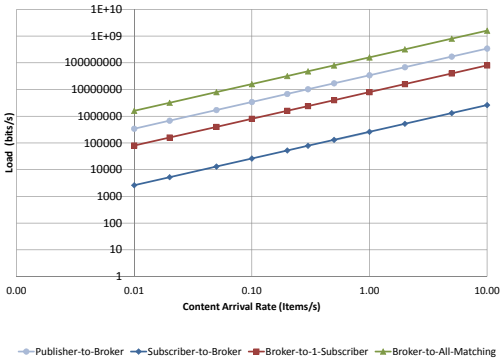

Fig. 12. Total traffic load in bits/sec between publisher, broker, and subscriber using OFSGP-Match

the broker in bits/s, W_s, the network throughput between the subscriber and the broker in bits/s, D, the content size in bytes, N_s, the number of subscriptions, R_p, the arrival rate of new content items for publication, and F_m, the average fraction of subscriptions that match an item. For our calculations, we assume that the publisher, broker, and subscriber perform only sequential (not parallel) computation.

We note that the metadata for each content item needs to be prepared once at the publisher. Then the prepared metadata needs to be blinded N_s times. We note that the time taken by the PRNG dominates the cost. For OFSGP-Match, N_s blinded sequences of length $n^{2\kappa+1}$ each are sent to the broker, along with the content item of size D bytes. In contrast, for the plaintext case, only the content item and n bits of metadata need to be sent.

Operation	Time (ms)
Metadata preparation at publisher	22
Random sequence at publisher	23
Blinding at publisher	4
Interest preparation at subscriber	10
Random sequence at subscriber	17
Blinding at subscriber	3
Evaluation at broker	1
End-to-end run	80

Table 2. Time taken for OFSGP-Match operations.

The broker needs to perform N_s matches for each new content item (which is small for the confidential match as measured, and negligible for the plaintext match). The broker must deliver the content item of size D bytes to subscribers corresponding to each of the F_m matching subscriptions in both the plaintext and confidential matching cases.

The subscriber needs to prepare each interest (subscription) once. For each match, the subscriber must generate R_p blinded sequences (each of length $1+n^{2\kappa+1}$ bytes) in every second and transmit them to the broker to keep up with the content arrival rate. The subscriber can choose to prepare and transmit a block of blinded sequences for multiple future matches. If content arrives too fast, if the subscriber has to manage a large number of subscriptions, or if the network between the subscriber and broker is slow, those factors will respectively dominate the cost at the subscriber.

Assuming subscriptions have been prepared and sent to the broker, we can calculate the end-to-end time from the arrival of the new content item at the publisher to its delivery to all subscribers with matching subscriptions if the parameters are known. We can also calculate the traffic load generated by the publisher, subscriber, and broker.

We consider a numerical example, with $n = 32$ and $\kappa = 1$ for which we have actual measurements; we let $W_p = W_s = 10^8$bits/s, $D = 10^6$ bytes, and $F_m = 0.2$. We vary N_s from 0 to 100 in steps of 10 and plot the end-to-end time in Figure 11. We note that the end-to-end time in the case of the plaintext publish/subscribe system as well as in the case of a confidential system based on OFSGP-Match is linear; the cost of confidential matching is higher by a small constant factor.

Now for the case $N_s = 100$, we vary R_p in the range 0.01 to 10, and we compute and plot the traffic load in bits between the publisher and the subscriber, the subscriber and the broker, the broker and each subscriber, and the total load from the broker to all subscribers in Figure 12. We use a logarithmic scale for both axes of the plot. We note that the load due to the delivery of matching content (which is the same for the plaintext case and the confidential case) is far greater than the overhead for confidential matching in our example. Since the exact complexity of OFSGP-Match is known, it is easy to calculate the performance for other values of the parametric model.

Based on this performance characterization, we can see that OFSGP-Match can support a confidential content-based publish/subscribe system in limited, but useful, settings. We note that the operations at the publisher, subscriber, and broker are all "embarassingly" parallel, which motivates the use of OFSGP-Match in cloud computing applications.

7 Conclusion

We showed how all predicate circuits in NC^1 (logarithmic-depth, bounded fan-in) can be confidentially computed by a broker while guaranteeing perfect information-theoretic security in a Confidential Content-Based Publish/Subscribe System. We presented three protocols—UGP-Match, FSGP-Match and OFSGP-Match—based on 2-decomposable randomized encodings of group programs for circuits in NC^1 .

On the theoretical front we achieved a level of expressivity that had not been previously attained and on the practical front we made a substantial advance towards a practical scheme with low-degree polynomial complexity. Our protocols have the benefit of being conceptually clean and simple, and we can capture their exact complexity without needing to employ the big-Oh notation, see Table 1.

We have made two implementations publicly available—one in JavaScript for delivery over the web [26] and another more capable solution in Java [27] for enterprise applications. Using a parametric model based on measurements from one of these implementations, we characterized the anticiapted performance of a confidential publish/subscribe system using OFSGP-Match relative to a cleartext baseline. A future parallel or distributed implementation can provide further speed-ups. An open question is how to design a faster protocol that can potentially scale to wireline speeds of edge networks, while preserving the expressivity and guarantees that OFSGP-Match provides.

References

[1] B. Applebaum. Randomly encoding functions: A new cryptographic paradigm - (invited talk). In *ICITS*, pages 25–31, 2011.

[2] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. In *FOCS*, pages 166–175, 2004.

[3] B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. In *IEEE Conference on Computational Complexity*, pages 260–274, 2005.

[4] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[5] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *ICDCS*, pages 262–272, 1999.

[6] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[7] A. Ben-David, N. Nisan, and B. Pinkas. Fairplaymp: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security*, pages 257–266, 2008.

[8] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[9] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[10] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[11] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[12] A. K. Datta, M. Gradinariu, M. Raynal, and G. Simon. Anonymous publish/subscribe in p2p networks. In *IPDPS*, page 74, 2003.

[13] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563, 1994.

[14] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[15] O. Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[16] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

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

[18] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: tool for automating secure two-party computations. In *ACM Conference on Computer and Communications Security*, pages 451–462, 2010.

[19] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[20] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In *STOC*, pages 433–442, 2008.

[21] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Extracting correlations. In *FOCS*, pages 261–270, 2009.

[22] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[23] A. Kerckhoffs. Kerckhoffs's principle. http://en.wikipedia.org/wiki/Kerckhoffs's_principle, 1883.

[24] V. Kolesnikov. Advances and impact of secure function evaluation. *Bell Labs Technical Journal*, 14(3):187–192, 2009.

[25] V. Kolesnikov and T. Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography*, pages 83–97, 2008.

[26] R. Krishnan. Illustration of OFSGP-match using Javascript and Scheme for Web delivery. http://cosocket.com/rajesh-krishnan/software/jsm/jsmdemo.html, 2013.

[27] R. Krishnan. Decision Evaluation in Encrypted Domains—OFSGP-Match Implementation in Java. https://github.com/Cosocket-LLC/deed, 2014.

[28] L. Malka. Vmcrypt: modular software architecture for scalable secure computation. In *ACM Conference on Computer and Communications Security*, pages 715–724, 2011.

[29] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[30] C. Raiciu and D. S. Rosenblum. Enabling confidentiality in content-based publish/subscribe infrastructures. In *SecureComm*, pages 1–11, 2006.

[31] T. Sander, A. L. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *FOCS*, pages 554–567, 1999.

[32] M. Srivatsa and L. Liu. Securing publish-subscribe overlay services with EventGuard. In *ACM Conference on Computer and Communications Security*, pages 289–298, 2005.

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

[34] E. Viola. Gems of theoretical computer science. Lecture no. 11. Barrington's theorem. http://www.ccs.neu.edu/home/viola/classes/gems-08/lectures/le11.pdf, 2009.

[35] Wikipedia. NC^1 - barrington's theorem. http://en.wikipedia.org/wiki/NC_(complexity)#Barrington.27s_theorem, 2012.

[36] Wikipedia. NC - complexity class. http://en.wikipedia.org/wiki/NC_(complexity), 2012.

[37] Wikipedia. Multiplexer. http://en.wikipedia.org/wiki/Multiplexer, 2012.

[38] Wikipedia. Notation for representing permutations. http://en.wikipedia.org/wiki/Permutation#Notation, 2012.

[39] Wikipedia. Solvable group. http://en.wikipedia.org/wiki/Solvable_group, 2012.

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

A Blinding Lemma

For the sake of completeness we repeat the statement of Lemma 4 and follow it up with the proof.

Lemma (Blinding Lemma)**.** *Given a sequence of group elements, S, of length L, the blinded sequence $\mathcal{BS}(S)$ has the following two properties:*
Preserves value*: $Value(\mathcal{BS}(S)) = Value(S)$*
Uniform distribution*: $\mathcal{BS}(S)$ is uniformly distributed over the space of all sequences of group elements of length L with the same value, i.e. for any sequence $S' = g'_1, g'_2 \dots g'_L$ we have that*

$$Pr(\mathcal{BS}(S) = S' \,|\, Value(S') = Value(S)) = \frac{1}{|G|^{L-1}}$$

where the probability is measured over the random choices of the blinders.

Proof. It is easy to see that blinding preserves the value of the sequence because the blinders, the r_i, cancel out when the elements of the blinded sequence are multiplied together.

Now, we need to show the uniform distribution property. First, observe that the space of all sequences S' such that Value(S) = Value(S') has size exactly $|G|^{L-1}$. This is because we can pick the first $L-1$ elements, $g'_1, g'_2, \dots, g'_{L-1}$ of S' arbitrarily from the group (in $|G|^{L-1}$ ways) but having done that then (because these elements are chosen from a group) there is exactly one value for the n-th element g_L namely $g_{L-1}^{-1} \cdot g_{L-2}^{-1} \cdot \ldots \cdot g_1^{-1} \cdot$Value($S$) such that Value($S'$) = Value($S$).

Hence, to compute $\Pr(\mathcal{BS}(S) = S')$ we just need to compute the probability that the first $L-1$ elements of the two sequences ($\mathcal{BS}(S)$ and S') match because the condition, that Value(S') = Value(S) along with the already proven fact that Value($\mathcal{BS}(S)$) = Value(S') implies that the L-th elements must automatically match if the first $L-1$ match.

Let $\mathcal{BS}(S) = b_1, b_2, \dots, b_L$. Then we have argued that

$$\begin{aligned}
&\Pr((b_1 = s'_1) \wedge (b_2 = s'_2) \wedge \dots \wedge (b_L = s'_L) | \\
&\qquad (\text{Value}(S') = \text{Value}(S))) \\
&= \Pr((b_1 = s'_1) \wedge (b_2 = s'_2) \wedge \dots \wedge (b_{L-1} = s'_{L-1}) | \\
&\qquad (b_L = s'_L)) \\
&= \Pr((b_1 = s'_1) \wedge (b_2 = s'_2) \wedge \dots \wedge (b_{L-1} = s'_{L-1})) \\
&= \Pr(b_1 = s'_1) \times \Pr((b_2 = s'_2) | (b_1 = s'_1)) \\
&\qquad \times \Pr((b_3 = s'_3) | (b_1 = s'_1) \wedge (b_2 = s'_2)) \\
&\qquad \times \dots \\
&\qquad \times \Pr((b_{n-1} = s'_{n-1}) | (b_1 = s'_1) \wedge (b_2 = s'_2) \wedge \dots \\
&\qquad \dots \wedge (b_{L-1} = s'_{L-1}))
\end{aligned}$$

But, by the definition of blinding, the above is

$$\begin{aligned}
&\Pr(r_1 = s_1^{-1} \cdot s'_1) \times \Pr(r_2 = s_2^{-1} \cdot s_1^{-1} \cdot s'_1 \cdot s'_2) \\
&\qquad \times \dots \times \Pr(r_{L-1} = \prod_{i=1}^{L-1} s_i^{-1} \prod_{i=}^{L-1} s'_i) \\
&= \frac{1}{|G|} \times \frac{1}{|G|} \times \dots \times \frac{1}{|G|} \\
&= \frac{1}{|G|^{L-1}}.
\end{aligned}$$

But this means that $\mathcal{BS}(S)$ is uniformly distributed over the space of all sequences of group elements with the same value since the space of all such sequences is of size exactly $|G|^{L-1}$, as has been argued earlier. □

B Barrington Transform

B.1 Barrington Transform

We present the proof of Theorem 7 based on the treatment in [34, 35]. Note that the statement of the theorem is true for any cycle and not just the specific cycle $\alpha = (23451)$.

We present the proof as a series of lemmas.

Lemma 26 (Rotator lemma)**.** *Given two cycles $\alpha, \beta \in S_5$ any rotator $\rho(\alpha, \beta)$ has the property that*

$$\alpha = \rho^{-1}(\alpha, \beta) \cdot \beta \cdot \rho(\alpha, \beta)$$

Proof. Let

$$\alpha = (\alpha_1, \alpha_2, \dots, \alpha_5)_{\text{cycle}}$$
$$\beta = (\beta_1, \beta_2, \dots, \beta_5)_{\text{cycle}}$$

Then, remember that their rotator is defined as

$$\rho(\alpha, \beta) := (\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, ..., \alpha_5 \to \beta_5).$$

Observe that the definition of the rotator depends on the representation-specific order of the elements of the cycles of α and β and thus there can be multiple rotators for the same α and β. For example, it is possible that representing α as $(\alpha_2, \alpha_3, \dots, \alpha_5, \alpha 1)_{\text{cycle}}$ could give rise to a different rotator $\rho(\alpha, \beta)$.

Though the rotator is representation-dependent and (potentially) not unique, nevertheless it is easy to see by explicit calculation that

$$\alpha = \rho^{-1}(\alpha,\beta)\cdot\beta\cdot\rho(\alpha,\beta)$$

□

Lemma 27 (The cycle does not matter)**.** *Let $\alpha,\beta \in S_5$ be two cycles. Let $f:\{0,1\}^n\to\{0,1\}$. Then f is α-computable with length ℓ if and only if f is β-computable with length ℓ.*

Proof. Suppose that $(g_1^0,...,g_\ell^0)(g_1^1,...,g_l^1)(k_1,...k_\ell)$ β-computes f; we claim that $(\rho^{-1}(\alpha,\beta)g_1^0, g_2^0,\ldots,g_\ell^0\rho(\alpha,\beta))$ $(\rho^{-1}(\alpha,\beta)g_1^1, g_2^1,\ldots,g_\ell^1\rho(\alpha,\beta))$ (with the same indices k_i) α-computes f. To see this, note that it follows from Lemma 26 that

$$\prod_{i=1}^{\ell} g_i^{x_{k_i}} = 1_{S_5} \Rightarrow$$

$$\rho^{-1}(\alpha,\beta)\prod_{i=1}^{l} g_i^{x_{k_i}}\rho(\alpha,\beta) = \rho^{-1}(\alpha,\beta)\cdot\rho(\alpha,\beta) = 1_{S_5},$$

$$\prod_{i=1}^{\ell} g_i^{x_{k_i}} = \beta \Rightarrow$$

$$\rho^{-1}(\alpha,\beta)\prod_{i=1}^{l} g_i^{x_{ki}}\rho(\alpha,\beta) = \rho^{-1}(\alpha,\beta)\cdot\beta\cdot\rho(\alpha,\beta) = \alpha.$$

□

Lemma 28 ($f \Rightarrow 1-f$)**.** *If $f:\{0,1\}^n\to\{0,1\}$ is α-computable by a group program of length ℓ, so is $1-f$.*

Proof. First apply the previous lemma to α^{-1}-compute f. Then multiply last group elements g_ℓ^0 and g_ℓ^1 in the group program by α. (Note that $\alpha^{-1} = (51234)$.) □

Lemma 29 ($f,g \Rightarrow f\wedge g$)**.** *If f is α-computable with length ℓ and g is β computable with length ℓ then $(f\wedge g)$ is $(\gamma(\alpha,\beta))$-computable with length 4ℓ.*

Proof. Remember that commutator $\gamma(\alpha,\beta) = \alpha\cdot\beta\cdot\alpha^{-1}\cdot\beta^{-1}$. Concatenate 4 programs: (α-computes f, β-computes g, α^{-1}-computes f, β^{-1}-computes g). (f(x)=1)$\wedge$ (g(x)=1)$\Rightarrow$ concatenated program evaluates to $(\alpha\cdot\beta\cdot\alpha^{-1}\cdot\beta^{-1} = \gamma(\alpha,\beta))$; but if either $f(x)=0$ or $g(x)=0$ then the concatenated program evaluates to 0. For example, if $f(x)=0$ and $g(x)=1$ then the concatenated program gives $1\cdot\beta\cdot 1\cdot\beta^{-1} = 1$. □

It only remains to see that we can apply the previous lemma while still computing with respect to a cycle. The following lemma critically relies on the non-solvability of S_5 [35].

Lemma 30. *$\exists\alpha,\beta$ cycles such that $\gamma(\alpha,\beta) = \alpha\cdot\beta\cdot\alpha^{-1}\cdot\beta^{-1}$ is a cycle.*

Proof. Let $\alpha := (23451)$, $\beta := (35421)$, we can check $\alpha\beta\alpha^{-1}\beta^{-1} = \gamma = (35214)$ is a cycle. □

The proof of the theorem follows by induction on d using previous lemmas.

B.2 $(\alpha,1)$-preserving Barrington Transform

The proof of Theorem 18 is basically the same as the proof of Theorem 7 with minor modifications to account for the fact that we are now transforming the circuit into an $(\alpha,1)$-preserving group program. We present the proof as a series of lemmas below.

Lemma 31 (The cycle does not matter)**.** *Let $\alpha,\beta \in S_5$ be two cycles, let $f:\{0,1\}^n\to\{0,1\}$. Then f is α-computable with length ℓ if an only if f is β-computable with length ℓ.*

Proof. The proof is similar to that of Lemma 27. Let $\alpha,\beta,\rho(\alpha,\beta) \in S_5$ be as in that lemma. Suppose that $(g_1,g_2,\ldots,g_{L+1})$ $(k_1,k_2,\ldots,k_L)$ β-computes f then it follows, in straightforward fashion that $(\rho^{-1}(\alpha,\beta)g_1,g_2,\ldots,g_{L+1}\rho(\alpha,\beta))$ $(k_1,k_2,\ldots,k_L)$ α-computes f. □

Lemma 32 ($f \Rightarrow 1-f$)**.** *If $f:\{0,1\}^n\to\{0,1\}$ is α-computable by a group program of length ℓ, so is $1-f$.*

Proof. First apply the previous lemma to α^{-1}-compute f. Then multiply the last group element g_L in the group program by α. □

Lemma 33 ($f,g \Rightarrow f\wedge g$)**.** *If f is α-computable with length ℓ and g is β computable with length ℓ then $(f\wedge g)$ is $(\gamma(\alpha,\beta) = \alpha\cdot\beta\cdot\alpha^{-1}\cdot\beta^{-1})$-computable with length 4ℓ.*

Proof. The proof is identical to that of Lemma 29. □

The proof of the theorem follows by induction on d using previous lemmas. and Lemma 30.