Proc. of ICDM 2006, Pages 149–164

# Applications of Orthogonal Arrays to Computer Science

K. Gopalakrishnan
*Department of Computer Science, East Carolina University, Greenville, NC 27858, U.S.A.*
Douglas R. Stinson
*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario–N2L 3G1, Canada.*

**Abstract.** Orthogonal arrays (OAs) are basic combinatorial structures, originally studied by statisticians motivated by their applications to design of experiments. In recent years, they have found numerous applications in computer science. Among their applications are **derandomization of algorithms, random pattern testing of VLSI chips, authentication codes, universal hash functions, threshold schemes,** and **perfect local randomizers**. In this article, we survey the applications of orthogonal arrays to computer science.
*keywords:* Orthogonal Arrays, Computer Science, Cryptography, Applications.

## 1 Introduction

The field of combinatorial design theory has gained a lot with the advent of computers. Many questions, which remained open for several years, about the existence of certain combinatorial designs have been settled by computer searches. A classic example is the proof of the non-existence of projective planes of order 10. However, more often than not, the contributions of combinatorial designs to computer science have been overlooked and not appreciated.

Recent years have seen numerous interesting applications of combinatorics to computer science in general and to cryptography in particular. This is exemplified by the following surveys: applications of error-correcting codes to cryptography (Sloane [45]), applications of combinatorial designs to computer science (Colbourn and Van Oorschot [14]), applications of finite geometry to cryptography (Beutelspacher [8]), applications of combinatorial designs to cryptography (Stinson [50]) and applications of combinatorial designs to communications, cryptography and networking (Colbourn et al., [15]). Thus the interaction between combinatorial designs and computer science is symbiotic in nature rather than parasitic.

There are a wide variety of combinatorial designs and in this article, we will focus exclusively on the applications of **orthogonal arrays**. The applications of orthogonal arrays to statistical design of experiments are well known and can even be found in textbooks (see Hedayat et al., [24] for example). Hence, we will focus exclusively on the applications of orthogonal arrays to computer science. Among their applications are **derandomization of algorithms, random pattern testing of VLSI chips, authentication codes, universal hash functions, threshold schemes,** and **perfect local randomizers**. The above collection of applications coming from diverse areas attests to the usefulness of orthogonal arrays.

In Section 3, applications of OAs to some sampling strategies associated with randomized algorithms are presented. Section 4 explains how the same techniques could be exploited in random pattern testing of VLSI chips in an effective manner. Under appropriate circumstances, a randomized algorithm can be transformed into a deterministic polynomial time algorithm using orthogonal arrays. This is known as total derandomization and it is the topic of Section 5.

Next we focus on applications of OAs to cryptography. Sections 6 and 7 discuss the applications of OAs to authentication codes and universal hash functions respectively. In Section 8, the construction of ideal secret sharing schemes for threshold access structures using orthogonal arrays is illustrated. Finally, perfect local randomizers, which are really orthogonal arrays in disguise, are discussed in Section 9.

# 2 Definitions

An *orthogonal array* $OA_\lambda(t, k, v)$ is a $\lambda v^t \times k$ array of $v$ symbols, such that in any $t$ columns of the array every one of the possible $v^t$ ordered $t$-tuples (not necessarily distinct) of symbols occurs in exactly $\lambda$ rows. Usually $t$ is referred to as the *strength* of the orthogonal array, $k$ is called the number of *factors*, $v$ is called the number of *levels* and $\lambda$ is called the *index* of the orthogonal array. If $\lambda = 1$, then we write $OA(t, k, v)$. An orthogonal array is said to be *simple* if no two rows are identical. Of course, an array with $\lambda = 1$ is simple. In this article, we consider only simple arrays. As an example, an $OA(3, 4, 2)$ is shown in Table 1. We refer the reader to the excellent book by Hedayat et al., [24] for the basic theory of orthogonal arrays including methods of constructions and bounds. In this survey, our primary focus will be on their applications.

Table 1: An $OA(3, 4, 2)$

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

# 3 Partial Derandomization of Monte Carlo Algorithms

## 3.1 Basics

There are two basic types of randomized algorithms viz., Las Vegas algorithms and Monte Carlo algorithms. Here we will be concerned primarily with the Monte Carlo algorithms. For any problem instance $I$, a Monte Carlo algorithm always gives an answer, but the answer may be incorrect with some probability $\epsilon$. Monte Carlo algorithms are usually assumed to be algorithms for decision problems, and we will also make that assumption here.

A Monte Carlo algorithm is defined to be *yes-biased* if the following conditions are satisfied:

1. If the instance $I$ is a no-instance, then the algorithm answers "no".

2. If the instance $I$ is a yes-instance, then the probability that the algorithm answers "yes" is at least $1 - \epsilon$, where $\epsilon \geq 0$ is some fixed constant.

Hence, if the algorithm answers "yes", then we know that the answer is correct. However, if the algorithm answers "no", then there is the possibility that the answer may be incorrect. A *no-biased* Monte Carlo algorithm is defined in the obvious way and there is no loss of generality in considering only yes-biased Monte Carlo algorithms.

A yes-biased Monte Carlo algorithm can be viewed as a two-stage procedure in which first a "sample point" is chosen at random and next a deterministic procedure is applied to the sample point. This view is usually known as the *witness paradigm*. In the generic case the sample point is an integer $r \in \{0, 1, \ldots, n-1\}$ and the deterministic procedure can be viewed as computing a zero-one valued function $f(I, r)$. If $f(I, r) = 0$ then the algorithm answers "no" and if $f(I, r) = 1$ then the algorithm answers "yes". The function $f$ should satisfy the following: If $I$ is a no-instance, then for all $r \in \{0, 1, \ldots, n-1\}$, $f(I, r) = 0$ and If $I$ is a yes-instance, then the fraction of the values of $r$ for which $f(I, r) = 1$ is at least $1 - \epsilon$.

There are many problems for which no fast deterministic algorithm is known, but which can be solved efficiently using randomized algorithms. Notable examples are the massive use of randomness in computational number theory [4] and in parallel algorithms [32]. A good survey of randomization techniques used in sequential and distributed algorithms is presented in [22].

## 3.2 Two-Point Based Sampling

The reason why a biased algorithm is useful is that the error probability can be made as small as desired by repeated application of the algorithm. Suppose we have a yes-biased algorithm, and we apply the algorithm $k$ times. If we get at least one "yes" answer we can quit, since the instance $x$ must be a yes-instance. On the other hand, if $x$ is a yes instance, then the probability of getting $k$ "no" answers is at most $\epsilon^k$, which can be made arbitrarily small by taking $k$ large enough.

However, the conclusion that the error probability can be made exponentially small is based on the assumption that the sequence of random numbers used in the successive trials are independent and uniformly distributed. When a randomized algorithm is implemented in actual practice, one always uses a pseudo-random number generator which produces a sequence whose later values come from the previous values in a *deterministic* fashion. This invalidates the assumption of independence and might cause one to regard results about randomized algorithms with suspicion.

In view of this objection, many researchers attempted to construct pseudo-random number generators for which provable bounds could be obtained on the probability that none of the $k$ successive values produced is a witness. A simple and effective method known as *two-point sampling* is developed by Chor and Goldreich.

For the sake of simplicity, we assume that $U = Z_p$, the set of residues modulo a prime $p$. If $n = |U|$ is not a prime, then we can use the smallest prime number $p$ such that $n \leq p \leq 2n$; that such a prime exists is proved by Chebyshev and later by Paul Erdös. The construction, developed by Chor and Goldreich [12] building on

an observation by Joffe [27], proceeds as follows: Choose two random independent elements $a$ and $b$ in $Z_p$ with uniform probability. Compute the residues $r_i = a + i \times b \bmod p$, for $1 \leq i \leq k$. These residues are then used as the random numbers in the $k$ trials of the Monte Carlo algorithm. It is not too hard to see that the $r_i$'s are pairwise-independent random variables. By using Chebyshev's inequality, Chor and Goldreich prove that the probability that none of the $r_i$'s is a witness is at most $\epsilon/(1-\epsilon)k$.

If the successive random numbers used are independent of each other, then the error probability is $\epsilon^k$ and the number of true random bits needed is $k \log n$. In the above technique, the number of random bits used is $2 \log n$ and the error probability achieved is $\epsilon/(1-\epsilon)k$. This is significant as random bits are a scarce resource and hence need to be used sparingly.

### 3.3 Relevance of Orthogonal Arrays

Orthogonal arrays could be used in two-point sampling strategies as follows. Consider an $OA(2, k, n)$. This array has $n^2$ rows. Generate $2 \log n$ random bits and use them to index a specific row of the OA. Now test the Monte Carlo algorithm on the $k$ points which are elements of the row selected.

A bound on the error probability can be calculated using elementary combinatorial properties of orthogonal arrays. We proceed to do this now. Let $U$ denote the universe of sample points and let $|U| = n$. Let $S \subseteq U$ be the set of witnesses or "good" sample points. Then $|S| = (1-\epsilon)n = m$ (say).

Let $x_i$ denote the number of rows of the orthogonal array in which there are exactly $i$ occurrences of the elements from $S$. Call a row of the matrix a *bad row* if none of the elements in the row is a witness. Then the error probability is simply the probability that the randomly selected row is a bad row. Hence, for a given $S$, the error probability is

$$err(S) = \frac{x_0}{n^2}.$$

Note that we actually do not know the set $S$ explicitly. The only thing that we know about $S$ is the fact that $|S| = m$. So the actual error probability is bounded above by

$$err = \max\{err(S) : S \subseteq U, |S| = m\}.$$

We first derive three simple equations from the elementary properties of orthogonal arrays. Since an $OA(2, k, n)$ has $n^2$ rows, we have

$$\sum_{i=0}^{n} x_i = n^2.$$

We can count the number of occurrences of good points in the array in two ways. There are exactly $x_i$ rows in which there are $i$ occurrences of good points. An $OA(2, k, n)$ is also an $OA_n(1, k, n)$. Hence in any column any point occurs exactly $n$ times. So, the number of occurrences of good points in a column is $nm$. As there are $k$ columns, the total number of occurrences of good points in the whole array is $knm$. This yields the following equation:

$$\sum_{i=0}^{n} i x_i = knm.$$

Once again, we can count the number of occurrences of pairs of good points which lie in the same row in two ways. In any row in which there are $i$ occurrences of good points, there will be $i(i-1)$ occurrences of pairs of good points. On the other hand, if we look at any two columns, the number of occurrences of pairs of good points which lie in the same row is simply $m^2$. This is because any particular pair of good points occurs only once in any given pair of columns since the array is an $OA(2,k,n)$, and there are $m^2$ different such pairs of good points. Now the two columns can be selected in $k(k-1)$ ways and so the total number of occurrences is $k(k-1)m^2$. This yields the following equation:

$$\sum_{i=0}^{n} i(i-1)x_i = k(k-1)m^2.$$

To compute the error probability, we need an upper bound on $x_0$. Equivalently, we need a lower bound on $\sum_{i=1}^{n} x_i$, subject to the constraints

$$\begin{aligned} \sum_{i=1}^{n} ix_i &= knm \\ \sum_{i=1}^{n} i(i-1)x_i &= k(k-1)m^2. \end{aligned}$$

Now, let $z$ be any real number. Then clearly

$$\begin{aligned} 0 &\leq \sum_{i=1}^{n}(i-z)^2 x_i \\ &= \sum_{i=1}^{n}(i^2 - 2zi + z^2)x_i \\ &= \sum_{i=1}^{n} i^2 x_i - 2z\sum_{i=1}^{n} ix_i + z^2 \sum_{i=1}^{n} x_i \\ &= k(k-1)m^2 + knm - 2zknm + z^2 \sum_{i=1}^{n} x_i \end{aligned}$$

It follows that

$$\sum_{i=1}^{n} x_i \geq \frac{2knmz - knm - k(k-1)m^2}{z^2}.$$

The right hand side of the above equation is maximized when

$$z = \frac{n + (k-1)m}{n}.$$

Hence we get

$$\sum_{i=1}^{n} x_i \geq \frac{kmn^2}{n + (k-1)m}.$$

Equivalently, we have

$$x_0 \leq n^2 - \frac{kmn^2}{n + (k-1)m}.$$

Hence we get the following bound on the error probability:

$$err \leq 1 - \frac{km}{n + (k-1)m}.$$

Since $m = n(1-\epsilon)$, we have that

$$err \leq \frac{\epsilon}{1 + (k-1)(1-\epsilon)}.$$

Chor and Goldreich's construction of sequence of a two-point based random numbers is essentially the construction of a specific $OA(2, k, n)$. Our technique presented above is based on an *arbitrary* $OA(2, k, n)$ and hence is more general. On the other hand, despite the generality of the technique, our bound of

$$\frac{\epsilon}{1 + (k-1)(1-\epsilon)}$$

is *always* less than or equal to their bound of

$$\frac{\epsilon}{(1-\epsilon)k}.$$

Also, while the proof of our bound uses only elementary combinatorial properties, the proof of their bound uses sophisticated machinery like Chebyshev's inequality. Thus we have a simpler proof of a better bound which is applicable in a more general context. This is due to the fact that two-point sampling is simply an application of orthogonal arrays.

In a slightly different context, Spencer (see [46] and Section 4) developed a bound on the error probability using the Bonferroni inequalities. His bound is

$$\frac{1}{1 + k(1-\epsilon)}.$$

Once again, despite the use of complex techniques, this bound is inferior to the above bound developed using orthogonal arrays.

Further details on the analysis of two-point based sampling can be found in [21]. An extension of the same technique to the analysis of $t$-point based sampling can be found in [17].

# 4 Testing VLSI Chips

## 4.1 Background

The basic testing problem is to determine an optimal testing procedure to identify digital systems which have become faulty because of physical failures. In most cases involving VLSI chips, one is not usually concerned with detecting the exact physical failure; what is desired is merely to determine the existence of (or absence of) any physical failure.

The large number and complex nature of physical failures dictates that a practical approach to testing should avoid working directly with physical failures. Often the problem is attacked by describing the effects of physical failures at some higher level. This description is called a fault model [35]. If the fault model accurately describes all the physical failures of interest, then one only needs to derive tests to detect all the faults in the fault model.

In what follows, the digital system chosen is combinational circuits. This is because, in spite of their simplicity, combinational circuits capture the complexity of the testing process. Further, the widespread use of design for testability techniques, such as Level-Sensitive Scan Design or LSSD [18], that transform a sequential circuit into a combinational one for testing purposes, has lent increasing importance to specialized methods for testing combinational circuits.

In what follows, the fault model selected is the classical *Single, permanent Stuck-at Fault (SSF)* model. In this model, testing is done to detect the capability of every line in the circuit to carry both a 0 and a 1 signal. A Single Stuck-at Fault in a circuit is a good model of various physical failures, such as a break in a diffusion line caused by an unintentional scratch on IC chip. Further the high coverage of multiple faults by Single Fault test sets has made the SSF model even more important [1, 35].

The basic testing problem, in the specific context of combinational circuits and SSF model, can be described as follows.

Let $C$ be a well formed combinational circuit [23] with $l$ lines and $F$ be the set of $2l$ Single Stuck-at Faults in $C$. Let $x_1, x_2, \ldots, x_n$ be the set of all primary inputs and $y_1, y_2, \ldots, y_m$ be the set of all primary outputs of $C$. The symbol $Z[X]$ is used to denote the output vector $Y = (y_1, y_2, \ldots, y_m)$, when the input vector $X = (x_1, x_2, \ldots, x_n)$ is applied to the circuit. Let $Z_f[X]$ represent the output vector of the circuit in the presence of the fault $f \in F$. It is clear that any input vector $X_t$ which produces different output vectors $Z[X_t]$ and $Z_f[X_t]$ is a test vector for $f$. The set of all test vectors which can detect $f$ will be denoted by $T(f)$. Thus

$$T(f) = \{X_t \mid Z[X_t] \neq Z_f[X_t]\}.$$

The basic testing problem is to construct a set of test vectors $T(F)$ such that for every fault $f \in F$, there exists at least one test vector in $T(F)$ which will detect $f$ i.e,

$$\forall\ f \in F\ \exists\ X_t \in T(F) \text{ such that } X_t \in T(f).$$

In practice, it is difficult to test for all the faults and hence what is attempted is merely a high amount of fault coverage (95% to 99%). The term *fault coverage* refers to the fraction of the modeled faults detected by the test vectors. Further the test costs may be broken down into two categories viz., cost of test generation and cost of test application. The former is a one-time cost and is measured by the computational effort required to generate the vectors. The later is a recurrent cost and is measured by the time taken to apply the vectors to the circuit under test.

## 4.2 Built-in Self Test

The traditional testing strategy is to compute a set $T(F)$ of input vectors such that for each possible fault there is an input vector in $T(F)$ that distinguishes the correct circuit from the circuit with the fault. If this strategy is used, the time required to fetch the patterns, from where they are stored to the circuit, severely limits the rate at which the patterns can be applied. Built-in self test (BIST) has been suggested as a way to overcome this bottleneck. Simply speaking, the idea is to have the chip test itself. The test vectors are generated by special purpose hardware which itself is a part of the chip. As a consequence the test vectors can be generated and applied as fast as one pattern per machine cycle. The speed gained is normally worth the additional investment in chip area.

Random pattern testing strategy is typically used in BIST schemes. In this approach, a random pattern generator is used to generate the test vectors which

are applied to the circuit under test. A comparator compares the actual response from the circuit under the test and the expected response to determine whether or not an error has been observed. The random test may stop either on the first observed error or after a sufficient number of random tests (test length) yield a correct response. Several random self-test architectures have been proposed in the literature [5]. Basically, all these versions use a linear feedback shift register (LFSR) to generate pseudo-random inputs that are fed to the circuit.

The fault coverage desired essentially dictates the test length in a random pattern test generator. So, let us focus on the estimation of test length [39] to achieve a prespecified amount of fault coverage. The detection probability $q_i$, of a fault $f_i$, is the probability that a randomly selected input vector will detect the fault. In other words

$$q_i = \frac{|T(f_i)|}{2^n}.$$

The problem of exact computation of detectability of a fault is NP-hard. However, algorithms which work well in practice for exact and approximate computations of detectability of each fault in a circuit are available [10, 40, 41]. The escape probability $e_i$, of a fault $f_i$, is the probability that the fault will go undetected after the application of $t$ random input vectors. The escape probability is given by

$$e_i = (1 - q_i)^t.$$

Similarly, the escape probability $e$ of a fault set $\{f_1, f_2, \ldots f_m\}$ is the probability that at least one member of the fault set will be left undetected after the application of $t$ random input vectors. The escape probability of a fault set can be computed using the Markov chain model (using the escape probabilities of individual faults). The minimum random pattern test length is then merely the smallest $t$ that satisfies $e \leq e_{th}$, where $e_{th}$ is the acceptable error probabaility. Finally, observe that this analysis has assumed independence of the random vectors generated.

## 4.3 Provably Good Pattern Generators

As with randomized algorithms, the primary objection to the random pattern testability is that the analysis of test length and escape probability assumed complete independence of the vectors generated and randomness of each vector. In practice we only use pseudo-random generators where the successive vectors produced are completely determined by the initial seed vector. Thus the analysis does not reflect reality and hence we cannot be really sure of the actual fault coverage provided.

In view of this objection, T. Spenser [46] developed a pseudo-random pattern generator for use in VLSI testing. This generator is essentially based on the idea of two-point sampling method for randomized algorithms developed by Chor and Goldreich [12]. The generator is implemented as a modification of the classical LFSR such that the increases in the delay and the required hardware are only by at most a small constant factor. Thus, it can be used in practice.

The input vectors can be considered as elements of the finite field $GF(2^n)$ with $2^n$ elements. Let $z$ be a generator of the cyclic multiplicative group of this field. The seed for the generator consists of an ordered pair $(a, b)$ of elements of $GF(2^n)$, such that $a \neq 0$. The $i$th element produced by the generator is $az^{i-1} + b$. The escape probability of a fault $f_i$ using this pattern generator is shown to be $1/(1+q_it)$ provided $q_it$ is an integer. The analysis of the escape probability is realistic and reflects the actual generator that is being used. Once again, this generator can be

interpreted as an orthogonal array $OA(2, t, 2^n)$ and hence the analysis developed in Section 3 is applicable here as well.

# 5 Total Derandomization

Any randomized algorithm typically uses random bits in the course of its computation. So, we somehow need to generate random bits to actually implement randomized algorithms. In this regard John Von Neumann says that "*anyone who attempts to generate random numbers by an arithmetical process is in a state of sin*". On the other hand, there exists chaotic processes in nature, such as radioactive decay and the thermal noise in a transistor, that allow the construction of a random number generator whose behavior is for all practical purposes equivalent to that of a true random source. Even in these cases whether the chaotic processes involved are truly random is a philosophical question related to the question of whether the universe is deterministic or not, and seems to be impossible to answer to everyone's satisfaction. For a philosophical discussion of randomness the reader is referred to the book by Von Mises [31] and the one by Popper [34]. In any case, it is clear that randomness should be considered as a scarce resource which should be used as meagerly as possible. Thus we could talk of *randomness complexity* like we do of time and space complexity. The randomness complexity of an algorithm could be defined as the function that describes the growth of the number of random bits used by the algorithm with the size of the input. Under this view we would like to make sure that the randomness complexity of our randomized algorithms is fairly small.

Hence many researchers have developed techniques to reduce the number of random bits needed in computation and to eliminate completely, if possible, the use of random bits. When we completely eliminate the random bits needed in the randomized algorithm, we are in effect converting the randomized algorithm to a deterministic algorithm. This process is called (total) derandomization. For an excellent survey of some derandomization techniques the reader is referred to the recent book by Alon and Spencer [3].

Our emphasis here will be on a derandomization technique known as the $t$-wise independence method. This method is a general technique for converting randomized algorithms, whose analysis depends only on $t$-wise rather than fully independent random choices, into deterministic ones. This method was first developed by M. Luby for the case $t = 2$ in the context of the maximal independent set problem [29] and later generalized to arbitrary constant $t$ by Alon, Babai and Itai [2].

The basic idea is to replace an exponentially large sample space by one of polynomial size. If a random variable on such a space takes a certain value with positive probability, then we can find a point in the sample space in which this happens in polynomial time simply by deterministically checking all the points. Moreover, the checking can be parallelized, if desired, using a polynomial number of parallel processors.

For the sake of simplicity, consider a randomized algorithm which requires $n$ random bits, where $n$ is also the size of the input. Then the size of the sample space is $2^n$. Suppose that the analysis of the randomized algorithm does not really require the complete independence of the random bits but only $t$-wise independence. Then, one could use any $OA_\lambda(t, n, 2)$ as the sample space instead of the above sample space with $2^n$ points.

If we can construct an $OA_\lambda(t, n, 2)$ with polynomial (in $n$) number of rows, then we have a polynomial sample space with $t$-wise independent random variables.

Such polynomial sample spaces exist for appropriate parameters is the content of the following theorem.

**Theorem 5.1.** *Suppose $n = 2^m - 1$ and $t = 2e$. Then there exists a symmetric probability space $\Omega$ of size $(n+1)^e$ and $t$-wise independent random variables $y_1, y_2, \ldots, y_n$ over $\Omega$, each of which takes the values 0 and 1 with equal probability.*

The above theorem can be proved by constructing an orthogonal array $OA_{2^{e(m-2)}}(2e, n, 2)$. Orthogonal arrays with these parameters can be constructed from BCH codes with designed distance $2e+1$ as explained in [2].

While this method is simple and powerful, it does not extend to the cases where $f(n)$-wise independence of the random variables is needed, where $f(n)$ is an increasing (however small the growth rate may be) function of $n$, as shown by the following theorem. This fact was first observed by Chor *et al* in [13].

**Theorem 5.2 ([13]).** *The size of the sample space constructed in Theorem 5.1 is asymptotically optimal and hence the size of a sample space with $n$ $t$-wise independent non-trivial random variables can be polynomial in $n$ only when $t$ is a constant.*

The proof is a straightforward application of Rao bounds (see [36]) for orthogonal arrays and can be found in [2] and [13].

# 6 Authentication Without Secrecy

Authentication codes were invented in 1974 by Gilbert, MacWilliams and Sloane [19] and have been extensively studied in recent years. The general theory of unconditional authentication was developed by Simmons (see e.g. [43]). In this section, we will give a brief review of some relevant known results and explain the relevance of orthogonal arrays in this context. For a general survey on authentication, the reader is referred to [44].

In the usual model for authentication, there are three participants: Alice, Bob, and an opponent (Oscar). Alice wants to communicate some information to Bob using a public communication channel. The purpose of authentication code is to protect the integrity of the information. That is, when Bob receives a message from Alice, he wants to be sure that the message was really sent by Alice and was not tampered with. However, Oscar has the ability to introduce messages into the channel and/or to modify existing messages. When Oscar places a (new) message into the channel, this is called *impersonation*. When Oscar sees a message $m$ and changes it to a message $m' \neq m$, this is called *substitution*.

We will use the following notation. An *authentication code without secrecy* is a four-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{K}, \mathcal{E})$, defined as follows:

1. $\mathcal{S}$ is a finite set of *source states* (a source state is analogous to a plaintext). Let $\mid \mathcal{S} \mid = k$.
2. $\mathcal{A}$ is a finite set of *authenticators*. Let $\mid \mathcal{A} \mid = l$.
3. $\mathcal{K}$ is a finite set of *keys*.
4. For each $K \in \mathcal{K}$, there is an *authentication rule* $e_K \in \mathcal{E}$. Each $e_K : \mathcal{S} \rightarrow \mathcal{A}$.

Alice and Bob will follow the following protocol. First, they jointly choose a secret key $K \in \mathcal{K}$. At a later time, Suppose Alice wants to communicate a source

state $s \in \mathcal{S}$ to Bob. Alice uses the authentication rule $e_K$ to produce the authenticator $a = e_K(s)$. The *message* $m = (s, a)$ is sent over the channel. When Bob receives $m$, he verifies that $a = e_K(s)$ to authenticate the source state $s$. Since each authentication rule is a function from $\mathcal{S}$ to $\mathcal{A}$, we can represent an authentication code by an $|\mathcal{E}| \times |\mathcal{S}|$ matrix, where the rows are indexed by authentication rules, the columns are indexed by source states, and then entry in row $e$ and column $s$ is $e(s)$. We call this matrix the *authentication matrix*

When Oscar performs impersonation or substitution, his goal is to have his bogus message $m' = (s', a')$ accepted as authentic by Bob, thus misleading Bob as to the state of the source. That is, if $e$ is the authentication rule being used (which is *not* known to Oscar), then he is hoping that $a' = e(s')$.

We assume that there is some probability distribution on $\mathcal{S}$, which is known to all the participants. Given this probability distribution, Alice and Bob will choose a probability distribution for $\mathcal{E}$, called an *authentication strategy.* Once Alice and Bob have chosen their authentication strategy, it is possible to compute, for $i = 0, 1$, a *deception probability* denoted by $P_{d_i}$, which is the probability that Oscar can deceive Bob by impersonation and substitution, respectively. In computing the deception probabilities, it is assumed that Oscar is using an optimal strategy.

It is not difficult to show that $P_{d_0} \geq 1/|\mathcal{A}|$ [43] and $P_{d_1} \geq 1/|\mathcal{A}|$ as well [47]. We are not only interested in minimizing the deception probabilities but also in minimizing the number of authentication rules, as this determines the amount of information to be exchanged between Alice and Bob in a secure fashion before they use the authentication code. Authentication codes (without secrecy) with minimum possible deception probabilities and the minimum number of encoding rules have been shown to be equivalent to orthogonal arrays by Stinson [48]. In the following result, the first part deals with codes in which $k \leq l+1$, while the second part deals with codes in which $k \geq l+1$.

**Theorem 6.1 ([48]).** *Suppose we have an authentication code without secrecy for $k$ source states and having $l$ authenticators, in which $P_{d_0} = P_{d_1} = 1/l$. Then*

1. *$| \mathcal{E} | \geq l^2$, and equality occurs if and only if the authentication matrix is an orthogonal array $OA(2, k, l)$ (with $\lambda = 1$) and the authentication rules are used with equal probability;*

2. *$| \mathcal{E} | \geq k(l-1)+1$, and equality occurs if and only if the authentication matrix is an orthogonal array $OA_\lambda(2, k, l)$ where*

$$\lambda = \frac{k(l-1)+1}{l^2},$$

*and the authentication rules are used with equal probability.*

# 7 Universal Hash Functions

Universal classes of hash functions were introduced by Carter and Wegman [11], and were studied further by Sarwate [38] and by Wegman and Carter [53]. Several applications of universal hash functions were discussed in [11] and [53]. The applications include complexity theory, randomized algorithms, associative memory, cryptography and authentication.

Let $A$ and $B$ be finite sets, and denote $a = |A|$ and $b = |B|$. We will assume that $a \geq b$. A function $h : A \to B$ will be termed a *hash function.* A finite set $H$ of

hash functions is $strongly - universal_2$ (or $SU_2$) if for every $x_1, x_2 \in A$ $(x_1 \neq x_2)$ and for every $y_1, y_2 \in B$,

$$|\{h \in H : h(x_1) = y_1, h(x_2) = y_2\}| = |H|/|B|^2.$$

For a motivation of this definition see [53]. For practical applications, it is also important that $|H|$ is small. This is because $\log_2 |H|$ bits are needed to specify a hash function from the family. It is fairly straightforward to show that strongly universal hash functions are equivalent to orthogonal arrays. The following theorem can be found in [51].

**Theorem 7.1.** *If there exists an $OA_\lambda(2, k, n)$, then there exists an $SU_2$ class $H$ of hash functions from $A$ to $B$, where $|A| = k$, $|B| = n$ and $|H| = \lambda n^2$. Conversely, if there exists an $SU_2$ class $H$ of hash functions from $A$ to $B$, where $a = |A|$ and $b = |B|$, then there exists an $OA_\lambda(2, k, n)$, where $n = b$, $k = a$ and $\lambda = |H|/n^2$.*

This theorem helps in establishing lower bounds on the number of hash functions and in constructing classes of hash functions which meet these bounds. It is straightforward to extend the definition and the theorem to $SU_t$ class of universal hash functions.

# 8 Threshold Schemes

In a bank, there is a vault which must be opened every day. The bank employs three senior tellers; but it is not desirable to entrust the combination to any one person. Hence, we want to design a system whereby any two of the three senior tellers can gain access to the vault, but no individual can do so. This problem can be solved by means of a threshold scheme.

Threshold schemes are actually a special case of secret sharing schemes. For an excellent survey of secret sharing schemes, the reader is referred to [49]. Threshold schemes were first described by Shamir [42] and Blakley [9] in 1979. Blakley's solution uses finite geometries, while Shamir's scheme is based on polynomial interpolation. Since then many constructions have been given for threshold schemes.

Informally a $(t, w)$-*threshold scheme* is a method of sharing a secret key $K$ among a finite set $\mathcal{P}$ of $w$ participants, in such a way that any $t$ participants can compute the value of $K$, but no group of $t-1$ (or fewer) participants can do so. The value of $K$ is chosen by a special participant called the *dealer*. The dealer is denoted by $D$ and we assume $D \notin \mathcal{P}$. When $D$ wants to share the key $K$ among the participants in $\mathcal{P}$, he gives each participant some partial information called a *share*. The shares should be distributed secretly, so no participant knows the share given to another participant.

At a later time, a subset of participants $B \subseteq \mathcal{P}$ will pool their shares in an attempt to compute the secret key $K$. If $| B | \geq t$, then they should be able to compute the value of $K$ as a function of the shares they collectively hold; if $| B | < t$, then they should not be able to compute $K$. In the example described above, we desire a $(2, 3)$-threshold scheme.

Often, we desire not only that an unauthorized subset of participants should be unable to compute the value of $K$ by pooling their shares, but also they should be unable to determine anything about the value of $K$. Such a threshold scheme is called a *perfect threshold scheme*. Here, we will be concerned only about perfect threshold schemes.

We will use the following notation. Let $\mathcal{P} = \{P_i : 1 \leq i \leq w\}$ be the set of participants. $\mathcal{K}$ is the *key set* (i.e., the set of all possible keys); and $\mathcal{S}$ is the *share*

*set* (i.e., the set of all possible shares). It is well known that in any perfect threshold scheme $| \mathcal{K} | \leq | \mathcal{S} |$. The schemes in which equality is attained are called *ideal threshold schemes*.

Orthogonal arrays come into picture once again by means of the following theorem due to E. Dawson, E.S. Mahmoodian and Alan Rahilly [16]. This theorem was first proved by Keith Martin in his Ph.D. thesis in 1992; However, only after several years it appeared in a paper by Jackson and Martin [26].

**Theorem 8.1 ([16],[26]).** *An ideal $(t, w)$ threshold scheme with $| \mathcal{K} | = v$ exists if and only if an $OA(t, w + 1, v)$ exists.*

The construction of the threshold scheme starting from the orthogonal array proceeds as follows. The first column of the OA corresponds to the dealer and the remaining $w$ columns correspond to the $w$ participants. To distribute a specific key $K$, the dealer selects a random row of the OA such that $K$ appears in the first column and gives out the remaining $w$ elements of the row as the shares. When $t$ participants later pool their shares, the collective information will determine an unique row of the OA (as $\lambda = 1$) and hence they can compute $K$ as the value of the first element in the row.

Can a group of $t - 1$ participants compute $K$ ? Any possible value of the secret along with the actual shares of these $t - 1$ participants determine a unique row of the OA. Hence, no value of the secret can be ruled out. Moreover, it is clear that the $t - 1$ participants can obtain no information about the secret.

# 9 Perfect Local Randomizers

A $(k, n)$ *sequence generator* stretches a binary random seed of length $k$ to a binary pseudorandom sequence of length $n$. A $(k, n)$ sequence generator is a $(k, n, e)$ *perfect local randomizer* (PLR) if every subset of $e$ of the $n$ binary output random variables is a set of $e$ independent and balanced random variables. The concept of perfect local randomizers was introduced and studied by Maurer and Massey [30]. They suggest the use of perfect local randomizers in key scheduling schemes where a small secret key must be stretched to a long key. It is obvious that a $(k, n, e)$ perfect local randomizer is equivalent to an $OA_{2^{k-e}}(e, n, 2)$. This equivalence was observed by Maurer and Massey. Thus perfect local randomizers are really orthogonal arrays in disguise. Hence, we will not further elaborate on them.

# 10 Concluding Remarks

In this article, we surveyed several applications of orthogonal arrays to computer science. The diverse areas from where the applications are drawn attests to the ubiquitous nature of orthogonal arrays.

Although, we have surveyed several applications of orthogonal arrays, our survey is by no means complete due to lack of space. We briefly mention some other applications. Orthogonal arrays are used in the decorrelation theory of block ciphers (see [52]) and in the construction of almost $k$-wise independent sample spaces (see [28]). They are also used in ramp schemes, which are generalizations of threshold schemes (see [25]). Ordered orthogonal arrays are a generalization of orthogonal arrays and they are used in constructions of $(t, m, s)$ nets (see [33]).

Orthogonal arrays are closely related to correlation-immune and resilient functions as well (see [20]). These functions are of fundamental importance in their own right and have a multitude of applications. Correlation-immune functions are used in stream ciphers as combining functions for running-key generators that are resistant to correlation attacks (see [37]). Resilient functions are used in the design of S-Boxes in block ciphers. Yet another application of resilient functions involves renewing a partially leaked cryptographic key, which is particularly relevant in the setting of quantum cryptography (see [6]). Finally resilient functions are also used in the generation of shared random strings in the presence of faulty processors. The last two applications were mentioned in [7] and [13].

# References


[1] V.K. Agarwal and A.S.F. Fung, Multiple Fault Testing of Large Circuits by Single Fault Test Sets, *IEEE. Trans. Comput.*, **C-30(11)** (1981).

[2] Noga Alon, Laszlo Babai and Alon Itai, A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem, *Journal of Algorithms*, **7** (1986), pp. 567–583.

[3] Noga Alon and Joel H. Spencer, *The Probabilistic Method*, John Wiley & Sons, Inc., (1992).

[4] E. Bach and J. Shallit, *Algorithmic Number Theory-Vol. I Efficient Algorithms*, MIT Press, (1996).

[5] Paul H. Bardell, William H. McAnney, and Jacob Savir, *Built-in Test for VLSI–Pseudorandom Techniques*, John Wiley & Sons, (1987).

[6] C. H. Bennett, G. Brassard, and A. K. Ekert, Quantum cryptography, *Scientific American*, **267(4)** pp. 26–33. (1992).

[7] C. H. Bennett, G. Brassard, and J. M. Robert, Privacy amplification by public discussion, *SIAM J. Computing*, **17** (1988), pp. 210–229.

[8] A. Beutelspacher, Applications of finite geometry to cryptography, in *Geometries, Codes and Cryptography* pp. 161–186. Springer-Verlag, 1990.

[9] G. R. Blakely, Safeguarding cryptographic keys, in *Proc. N.C.C., AFIPS Conference Proceedings*, **48** (1979), pp. 313–317.

[10] F. Brglez, P. Powrall, and R. Hung, Applications of testability analysis from ATPG to critical delay path tracing, in *Proc. of International Test Conference*, pp. 705–712. 1984.

[11] L. Carter and M. Wegman, Universal Hash Functions, *J. Computer and System Sciences*, **18** (1979), pp. 143–154.

[12] Benny Chor and Oded Goldreich, On the Power of Two-Point Based Sampling, *Journal of Complexity*, **5** (1989), pp. 96–106.

[13] Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky, The Bit Extraction Problem or $t$-Resilient Functions, *IEEE Symposium on Foundations of Computer Science*, **26** (1985), pp. 396–407.

[14] C. J. Colbourn and P. C. Van Oorschot, Applications of combinatorial designs in computer science, *ACM Computing Surveys*, **21** (1989), pp. 223–250.

[15] C.J. Colbourn, J.H. Dinitz, and D.R. Stinson, Applications of Combinatorial Designs to Communications, Cryptography and Networking, *Surveys in Combinatorics*, pp 37–100, Cambridge University Press, (1999).

[16] E. Dawson, E. S. Mahmoodian, and Alan Rahilly, Orthogonal arrays and ordered threshold schemes, *Australasian J. of Combinatorics*, **8** (1993), pp. 27–44.

[17] P. Dukes and A. C. H. Ling, A combinatorial error bound for t-point based sampling, *Theoretical Computer Science*, **310** (2004), pp. 479–488.

[18] E. B. Eichelberger and T. W. Williams, A Logic Design Structure for LSI Testability, in *Proc. of the 14$^{th}$ Design Automation Conference*, June 1977.

[19] E. N. Gilbert, F. J. MacWilliams, and N. J. A. Sloane, Codes which detect deception, *Bell System Tech. Journal*, **53** (1974), pp. 405–424.

[20] K. Gopalakrishnan and D. R. Stinson, Three characterizations of non-binary correlation immune and resilient functions, *Designs, Codes and Cryptography*, **5** (1995), pp. 241–251.

[21] K. Gopalakrishnan and D. R. Stinson, A simple analysis of the error probability of two-point based sampling, *Information Processing Letters*, **60** (1996), pp. 91–96.

[22] Rajiv Gupta, Scott A. Smolka, and Shaji Bhaskar, On randomization in sequential and distributed algorithms, *ACM Computing Surveys*, **26(1)** (1994), pp. 7–86.

[23] John P. Hayees, *Computer Architecture*, McGraw Hill, (1978).

[24] A. S. Hedayat, N. J. A. Sloane, and J. Stuffken, *Orthogonal Arrays: Theory and Applications*, Springer-Verlag, New York (1999).

[25] W. A. Jackson and K. Martin, A combinatorial interpretation of ramp schemes, *Australasian Journal of Combinatorics*, **14** (1996), pp. 51–60.

[26] W.A. Jackson and K. Martin, Combinatorial models for perfect secret sharing schemes, *Journal of Combinatorial Mathematics and Combinatorial Computing*, **28** (1998), pp. 249–265.

[27] A. Joffe, On a set of almost deterministic $k$-independent random variables, *Ann. Probab.*, **2(1)** (1974), pp. 161–162.

[28] K. Kurosawa, T. Johannsson, and D. R. Stinson, Almost $k$-wise independent sample spaces and their cryptologic applications, *Journal of Cryptology*, **14** (2001), pp. 231–253.

[29] Michael Luby, A Simple Parallel Algorithm for the Maximal Independent Set Problem, *SIAM J. Comput.*, **15(4)** (1986), pp. 1036–1053.

[30] U. M. Maurer and J. L. Massey, Local randomness in pseudorandom sequences, *Journal of Cryptology*, **4(2)** (1991), pp. 135–149.

[31] R. Von Mises, *Probability, Statistics and Truth*, Macmillan, New York (1957).

[32] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani, Matching is as easy as matrix inversion, *19$^{th}$ STOC*, pp. 345–354, (1987).

[33] Harald Niederreiter, Constructions of $(t, m, s)$ nets and $(t, s)$ sequences, *Finite Fields and Their Applications*, **11** (2005), pp. 578–600.

[34] K. R. Popper, *The Logic of Scientific Discovery*, London, (1959).

[35] D. K. Pradhan, *Fault Tolerant Computing: Theory and Techniques, volume 1*, Prentice Hall, Englewood Cliffs, NJ (1984).

[36] C. R. Rao, Factorial experiments derivable from combinatorial arrangements of arrays, *J. Royal Stat. Soc.*, **9** (1947), pp. 128–139.

[37] Rainer A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, (1986).

[38] D. V. Sarwate, A note on universal classes of hash functions, *Information Processing Letters*, **10** (1980), pp. 41–45.

[39] J. Savir and P. H. Bardell, On Random Pattern Test Length, *IEEE. Trans. Comput.*, **C-33(6)** (1984), pp. 467–474.

[40] Jacob Savir, Gary S. Ditlow, and Paul H. Bardell, Random Pattern Testability, *IEEE Trans. Computers*, **C-33(1)** (1984), pp. 79–90.

[41] S. C. Seth, B. B. Bhattacharya, and V. D. Agrawal, An exact analysis for efficient computation of random pattern testability in combinational circuits, *Digest of papers, FTCS-16*, (1986), pp. 318–323.

[42] A. Shamir, How to share a secret, *Comm. ACM.*, **22** (1979), pp. 612–613.

[43] G. J. Simmons, Message authentication: a game on hypergraphs, *Congressus Numerantium*, **45** (1984), pp. 161–192.

[44] G. J. Simmons, A survey of information authentication, *Contemporary Cryptology, The Science of Information Integrity*, pp. 379–419, IEEE Press, (1992).

[45] N. J. A. Sloane, Error-correcting codes and cryptography, *The Mathematical Gardner*, pp. 346–382, Prindle, Weber and Schmidt, (1981).

[46] Thomas H. Spencer, Provably Good Pattern Generators for a Random Pattern Test, *Algorithmica*, **11(5)** (1994), pp. 429–442.

[47] D. R. Stinson, The combinatorics of authentication and secrecy codes, *Journal of Cryptology*, **2** (1990), pp. 23–49.

[48] D. R. Stinson, Combinatorial characterizations of authentication codes, *Designs, Codes and Cryptography*, **2** (1992), pp. 175–187.

[49] D. R. Stinson, An explication of secret sharing schemes, *Designs, Codes and Cryptography*, **2** (1992), pp. 357–390.

[50] D. R. Stinson, Combinatorial Designs and Cryptography, *K. Walker, editor,* Surveys in Combinatorics, pp. 257–287. Cambridge University Press, (1993).

[51] D. R. Stinson, Combinatorial techniques for universal hashing, *Journal of Comp. and Sys. Sci.*, **48(2)** (1994), pp. 337–346.

[52] Serge Vaudenay,Decorrelation: A theory for block cipher security,*Journal of Cryptology*, **16** (2003), pp. 249–286.

[53] M. N. Wegman and J. L. Carter, New hash functions and their use in authentication and set equality, *J. Computer and System Sciences*, **22** (1981), pp. 265–279.