Group Testing and Batch Verification

Gregory M. Zaverucha and Douglas R. Stinson
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo ON, N2L 3G1, Canada
{gzaveruc, dstinson}@uwaterloo.ca

May 27, 2009

Abstract

We observe that finding invalid signatures in batches of signatures that fail batch verification is an instance of the classical group testing problem. We present and compare new sequential and parallel algorithms for finding invalid signatures based on group testing algorithms. Of the five new algorithms, three show improved performance for many parameter choices, and the performance gains are especially notable when multiple processors are available.

1 Introduction

A *batch verification algorithm* for a digital signature scheme verifies a list of n (message, signature) pairs as a group. It outputs 1 if all n signatures are valid, and it outputs 0 if one or more are invalid. In the most general case, the messages and signers may be different. Batch verification algorithms may provide large gains in efficiency, as verification of the n signatures is significantly faster than n individual verifications. In this paper, we address the problem of handling batches which fail verification, i.e., finding the invalid signatures which caused the batch to fail.

We observe that finding invalid signatures in bad batches is an instance of the *group testing problem*, which in brief, is as follows. Given a set B, of n items, d of which are defective, determine which items are defective by asking queries of the form "Does $B' \subseteq B$ contain a defective item?". Group testing is an old, well-studied problem, for which many algorithms exist. We re-cast some solutions to the group testing problem as solutions to the invalid signature finding problem, which are then compared for efficiency, parallelizability and accuracy.

In total, five new algorithms for finding invalid signatures are presented and included in our comparison. Of these, three give performance improvements. With a single processor, generalized binary splitting gives a modest improvement over the well-known binary splitting algorithm. In the case of two or more processors, large improvements are possible using one of two new group testing-based algorithms: Li's s-stage algorithm and the Karp, Upfal and Wigderson algorithm. The other two algorithms also have interesting properties. The algorithm based on cover-free families is fully parallelizable, and is an improved instance of a known algorithm, the id-code algorithm (for some parameter choices). The random matrices algorithm is probabilistic, fully parallelizable and enjoys a simple implementation.

We also give some general results on the limits of group testing that are also interesting in the context of finding invalid signatures in batches, such as the conditions when the naïve testing strategy is optimal.

Contributions and Outline After describing the link between finding invalid signatures in bad batches and group testing (§1.1, 1.2) we present new algorithms for finding invalid signatures based on techniques from the group testing literature (§2). These are grouped according to the adaptive (i.e. sequential §2.2) or nonadaptive (i.e. parallel §2.3) nature of the algorithm. We then compare the performance of the new algorithms (and some previously known algorithms) and determine the best one under various parameter choices (§3). For many parameter choices, especially with multiple processors, the new methods outperform known methods.

1.1 Batch Verification

Let the algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ specify a signature scheme. Gen takes as input a security parameter k, and outputs a signing and verification keypair (sk, pk). $\mathsf{Sign}(sk, m)$ outputs a signature σ on the message m using the secret key sk, and $\mathsf{Verify}(pk, \sigma, m)$ outputs 1 if σ is a valid signature of m under the secret key sk which corresponds to pk, and 0 otherwise.

Here is the most general definition of batch verification.

Definition 1.1 ([8])**.** Let $P_1, \ldots, P_n$ be n signers, with corresponding keypairs $K = \{(sk_1, pk_1), \ldots, (sk_n, pk_n)\}$ output by $\mathsf{Gen}(k)$ for some security parameter k. Let B be a list containing K, and n tuples of the form (P_{t_i}, σ_i, m_i) called the *batch* (note that the t_i and m_i values may be repeated.) The algorithm $\mathsf{Batch}(B)$ is a *batch verification algorithm* provided $\mathsf{Batch}(B) = 1$ if and only if $\mathsf{Verify}(pk_{t_i}, \sigma, m_i) = 1$ for all i.

A few variations appear in the literature, including the case with a single signer or the case of multiple signers with a single message. We also mention the related concept of *aggregate signatures*. Suppose $\sigma_1, \ldots, \sigma_n$ are signatures on messages $m_1, \ldots, m_n$ with corresponding verification keys $pk_1, \ldots, pk_n$. An aggregation algorithm is a public algorithm, which given the σ_i, m_i and pk_i $(i = 1, \ldots, n)$ outputs a compressed signature σ. An associated verification algorithm verifies if σ is a valid compressed signature, given pk_i and m_i (for $i = 1, \ldots n$).

A number of signature schemes in the literature support batch verification. Batch cryptography was introduced by Fiat [18, 19] to improve efficiency of an RSA-like scheme, where large numbers of operations are performed at a central site. History shows that secure batch verification algorithms are tricky to construct; a number of schemes were presented and subsequently broken or shown to be otherwise flawed. One example is the scheme of Al-Ibrahim et al. [1], which was broken by Stinson in [39]. Camenisch et al. list and reference ten proposed schemes which were later broken [8, §1.2]. Despite this poor track record, a number of signature schemes have batch verification, many of them based on the general techniques described in Bellare et al. [4].

We list a few examples, but omit details since the techniques in this work will apply to any scheme with batch verification. RSA* is an RSA-variant with batch verification presented by Boyd and Pavlovski [6]. DSA** is a signature scheme based on DSA, given by Naccache et al. [27], which uses the small exponents test from [4]. Camenisch et al. [8] give a variant of the Camenisch-Lysyanskaya signature scheme [7] which supports batch verification, present a batch verifier for the Π-IBS scheme of Chatterjee and Sarkar [11], and discuss batch verification of BLS signatures [5].

Practical considerations and implementation timings of batch verification are given in Ferrara et al. [17].

1.2 Finding Invalid Signatures in Bad Batches

Suppose we are given a batch B such that $\mathsf{Batch}(B) = 0$. We know that B contains at least one invalid signature, but what is the best way to determine *which* of the signatures do not verify? Verifying each signature individually is certainly an option, but can Batch be applied to subsets of B to perform less work overall? This problem can be considered the computational version of the batch verification problem (which is a decision problem). We name it the *invalid signature finding* (ISF) problem. This does not apply to aggregate signatures, where, since the batch is compressed, we do not have enough information to determine which of the original signatures were invalid.

We will treat the algorithm Batch as a generic test for invalid signatures, and present solutions which work for *any signature scheme* equipped with a Batch function as described in Definition 1.1. There are several advantages of generic ISFs.

1. *Applicability.* A generic ISF may be used with any signature scheme which provides batch verification. This includes future schemes.

2. *Implementation.* A single implementation may be used to locate bad signatures of multiple signature schemes, reducing the need to maintain multiple ISF implementations. The single generic ISF may be optimized, verified and otherwise improved since the effort is amortized over a larger number of applications.

3. *Ability to handle variations of the ISF problem.* The group testing literature has considered many variations of the problem, many of which are applicable to variations of the ISF problem.

The performance of an ISF may be evaluated based on the number of calls to Batch and the parallel performance of the algorithm.

1.2.1 Related Work

There have been five papers addressing the ISF problem. The first two are by Pastuszak et al. [30, 29]. They consider a generic Batch function for a signature scheme and study the *divide-and-conquer* method of finding bad signatures in [29]. The divide-and-conquer verifier was originally described in [27] under the name *cut and choose*, and is referred to *binary splitting* in the group testing literature. In brief, a batch B is divided in half, then Batch is recursively called on each sub-batch, until 1 is output (this sub-batch contains only valid signatures) or until the sub-batch has size one, which identifies the bad signatures. This method was implemented in the work of Ferrara et al. [17], and we discuss their findings in §2.2.1 when we relate the divide-and-conquer verifier to well-known techniques from group testing.

The second paper [30] approaches the problem using identification codes (id-codes), a code which encodes an ISF algorithm, by specifying subsets of B to test with Batch in such a way that all bad signatures may be identified. This approach is an instance of well-known non-adaptive group testing algorithms based on cover-free, separable and disjunct matrices, discussed in §2.3.1. A limitation of [30, 29] is that either the number of bad signatures in a batch, or a bound on the number of bad signatures is required *a priori*. This is common to most group testing algorithms as well.

The work of Law and Matt [24] improves the divide-and-conquer method by considering the details of the signature scheme. The second part of [24] gives an improved invalid signature finder using a special version of Batch. The batch verification and invalid signature finding tasks are combined, to allow information and intermediate computations from the verification step to be used in the ISF step. This trades off general applicability for improved computational efficiency. Along similar lines, Matt improves the performance of these methods when the number of invalid signatures is large [25]. This addresses a limitation of [24]. The improved techniques of [25] are applicable to the Cha-Cheon signature scheme [10] and the pairing-based schemes discussed in Ferrara et al. [17].

2 Group Testing-Based ISF Algorithms

We begin with a general description of the group testing problem called the (d, n)-*problem*. Consider a set of n items which contains exactly d defective items, called the *defective set*. Identification of a defective item requires the application of an error-free test, and we may test an arbitrary subset of the items. The test outcome may be *positive* if the subset of items contains at least one defective item, or *negative* if no defective items are present in the subset. An algorithm A which finds all d defective items is a solution to the problem. An algorithm where the tests are applied sequentially, and subsequent tests depend on the results of previous tests is called an *adaptive algorithm*. *Nonadaptive algorithms* require all tests to be specified at the outset; hence they may be executed in parallel.

Group testing has a long history, originating in World War II, motivated by the task of testing blood samples of draftees to detect syphilis [14, 16]. In this application, a single test on a combination of blood samples will return positive if any of the samples would test positive for syphilis. Since there were only a few thousand cases of the disease in millions of draftees, large subsets would come back negative, saving many individual tests. Group testing later found many industrial applications, a line of research initiated by Sobel and Groll [40]. In the past 50 years or so, a large literature has grown around the problem, and many variants have been considered. The book of Du and Hwang [15, 16] is a comprehensive reference.

It should now be clear that the ISF problem is a group testing problem: the items are signatures, the test applied to subsets is the batch verification algorithm, and the defectives are invalid signatures. This basic model makes the following assumptions:

- The subset tests all have the same cost, regardless of the number of items being tested.
- The number of defectives d, or a bound on d, is known *a priori*.

The first is a simplifying assumption, since the cost of $\mathsf{Batch}(B)$ is typically composed of a fixed overhead cost independent of $|B|$, plus a variable cost which grows with $|B|$. It does however, allow us to keep our analysis general, and ignore the details of Batch. The second allows some group testing algorithms to be more efficient. We will discuss the importance of the bound on d for each algorithm, and the behaviour of the algorithm when d is bounded incorrectly.

Probabilistic group testing (PGT) assumes a probability distribution on the defective set, while combinatorial group testing (CGT) does not. The only information CGT assumes about the defective set is that it is a d-subset of the n items. Some applications of batch verification may benefit from PGT if it is reasonable to make an assumption about the distribution of invalid signatures; however, we do not consider PGT algorithms in this paper.

Denote the minimal number of calls to Batch required to find d invalid signatures in a batch of size n by $M(d,n)$. First note that $M(d,n) \leq n-1$, by verifying $n-1$ signatures individually and inferring the validity of the last signature from knowledge of d and the other $n-1$ signatures. The following general lower bound is proven in [16, Cor. 2.1.11].

Theorem 2.1. $M(d,n) \geq \min\left\{n-1, 2\ell + \left\lceil \log \binom{n-\ell}{d-\ell} \right\rceil\right\}$ *for* $0 < \ell \leq d < n$.

Unless stated otherwise, $\log x$ is the base two logarithm of x, $\ln x$ is the natural logarithm of x, and e is the natural base.

2.1 Individual Testing

The simplest way of identifying all invalid signatures in a bad batch is to individually verify each signature. The question is, when is this naïve testing strategy optimal? Recall that $M(d,n)$ is the smallest possible number of tests for any (d,n) algorithm. Combining [16, Th. 3.5.1] and [16, Th. 3.5.3], we have the following result.

Theorem 2.2. *Let d be the number of invalid signatures in a batch of size n, and let $M(d,n)$ be as defined above. Then*

$$M(d,n) < n-1 \text{ for } n > 3d, \text{ and}$$
$$M(d,n) = n-1 \text{ for } n \leq 2.625d.$$

Therefore, when the number of bad signatures is at most $n/3$ it is possible to do better than individual testing, and when there are more than $n/2.625$ bad signatures the naïve strategy is optimal. What is best when $n < 3d$ and $n \geq 2.625d$ remains unknown; however, Hu, Hwang and Wang [21] conjectured that individual testing is optimal whenever $n \leq 3d$.

We note that individual testing is trivially parallelizable.

2.2 Adaptive ISF Algorithms

In this section we will present some adaptive ISF algorithms, based on group testing algorithms. In adaptive (or sequential) algorithms, the results of each test determines the items to be tested in subsequent tests. We will use the notation (d,n), where d is an upper bound on the number of bad signatures in the batch of size n.

2.2.1 Binary Splitting

An adaptive group testing algorithm is naturally represented as a binary tree. Nodes of the tree contain elements to be tested, starting at the root, which contains all n items. In binary splitting, at each level of the tree, we halve (i.e. divide as evenly as possible) the set of items in the parent node, to create two child nodes. When a test returns negative, this node becomes a leaf, since we know the set of items at this node is valid. Repeating this process recursively, we ultimately end up with nodes containing a single item, thus identifying the invalid items of the batch. By using depth first search from the root of the tree we may locate an invalid item using at most $\lceil \log(n) \rceil$ tests. We may remove the invalid item, and repeatedly apply the binary splitting algorithm to find d invalid items using at most $d\lceil \log(n) \rceil$ tests.

An implementation of binary splitting for the BLS signature scheme [5] is discussed in the work of Ferrara et al. [17]. They performed experiments with $n = 1024$ and they found binary splitting was faster than individual verification when $d < 0.15n$. In these experiments, a random fraction of the batch was corrupted, however Ferrara et al. note that in practice if corrupted signatures occur in bursts, the binary splitting algorithm will have better performance. Ordering of the batch may be an important consideration for applications using binary splitting.

A variant of binary splitting is Hwang's *generalized binary splitting*. The intuition of the algorithm is that there is roughly one defective item in every n/d items, and therefore a group smaller than $n/2$ could be tested and a defective found with fewer tests. When $d = 1$ the number of tests required by generalized binary splitting is $\lfloor \log(n) \rfloor + 1$, and when $d \geq 2$, the number of tests is not more than $d - 1 + \left\lceil \log \binom{n}{d} \right\rceil$, which gives a noticeable saving as d gets larger [16, Cor. 2.2.4].

Karp, Upfal and Wigderson describe an algorithm to identify a single invalid item using p processors in at most $\lceil \log_{p+1} n \rceil$ parallel tests [22]. The algorithm is identical to binary splitting when $p = 1$, since it uses a $(p + 1)$-ary tree in the same way that binary splitting does. At each level, p of the child sets are tested in parallel, and (if necessary) the validity of the $(p + 1)$-th set is inferred. We may repeatedly apply this algorithm to identify d invalid items in at most $d \lceil \log_{p+1} n \rceil$ parallel tests. We will refer to this algorithm as the *KUW algorithm*.

2.2.2 Li's s-Stage Algorithm

This algorithm has s rounds of testing, identifying good items at each round, until the last round when the algorithm corresponds to individual testing. Li's algorithm begins by grouping the batch into g_1 groups of size k_1 (some groups might have $k_1 - 1$ items). The groups are tested, and items in valid groups are set aside. The i-th stage divides the remaining elements into g_i groups of size k_i, tests them, and then removes items in valid groups. The final stage has $k_s = 1$, and remaining items are identified as valid or invalid.

When optimal choices (see [16, §2.3]) are made for g_i, k_i and s, the number of tests is not more than

$$\frac{e}{\log e} d \log \left(\frac{n}{d} \right) \ .$$

When p processors are available, Li's algorithm may be parallelized (see [15, p. 33]) and the number of parallel tests is not more than

$$\frac{e}{\log e} \frac{d}{p} \log \left(\frac{n}{dp} \right) + \ln \left(\frac{n}{dp} \right) + d \ .$$

2.3 Nonadaptive Algorithms

As we have seen, some adaptive algorithms are somewhat parallelizable. All nonadaptive algorithms are completely parallelizable. Recall that nonadaptive tests may be completely specified without information from previous tests. This can be be especially useful for online batch verification in a system with time constraints where a batch of n signatures arrive every time interval and must be processed before the next batch arrives, with a known number of tests. This might be applicable in the example of public key authentication in vehicular networks (this example is discussed in [8, 17]) or authentication of data reported periodically from sensors (as discussed in [9]). We continue to use the (d, n) notation defined at the beginning of Section 2.

2.3.1 Nonadaptive Group Testing with Cover-Free Families

A useful combinatorial structure for designing nonadaptive CGT (NACGT) algorithms is a cover-free family. Cover-free families are also studied under the terms *disjunct matrices* [16], *binary superimposed codes* [23], and *strongly selective families* [12]. Stinson et al. [37] discusses relations between these structures. We choose the language of cover-free families since they have found multiple applications in cryptography (see [20, 26, 36] for examples).

Definition 2.3. A *d-cover-free family* is a $t \times n$ binary matrix, with $n \geq d+1$, such that for any set of columns C and single column c such that $|C| = d$ and $c \notin C$ the following property holds. Let $U(C)$ be the binary OR of the columns in C. The cover-free property ensures that $c \notin U(C)$, that is, c is 1 in at least one position where $U(C)$ is 0. We will use the notation d-CFF(t, n) for cover-free families.

The cover-free property ensures that no d-set of columns "covers" any other column. A *d-separable* matrix satisfies a weaker property, namely, the OR of any two sets of d columns are distinct. While any d-separable matrix yields a NACGT algorithm, it is not efficient [16, Ch. 7]. We now describe how a d-CFF(t, n) defines an efficient (d, n) NACGT algorithm.

Input: Signatures $\sigma_1, \ldots, \sigma_n$, batch verification function Batch.
Output: Up to d invalid signatures.

1. Construct a matrix A which is a d-CFF(t, n) .
2. Associate σ_i to column i of A. Each row of A will define a sub-batch to test; if σ_i has a 1 in row j then σ_i is included in sub-batch j.
3. Compute $\mathsf{Batch}(B_1), \ldots, \mathsf{Batch}(B_t)$ where $B_i = \{\sigma_j : A_{i,j} = 1\}$.
4. For each row i such that $\mathsf{Batch}(B_i) = 1$ mark all $\sigma_j \in B_i$ as valid.
5. Output all the remaining signatures as invalid, i.e., signatures which do not belong to a valid batch.

We now explain how the algorithm correctly identifies valid signatures (and thus correctly outputs invalid signatures in step 5). Suppose σ_i is a valid signature. Let C be the set of columns corresponding to the invalid signatures. We are assuming that $|C| \leq d$. Let C' be any set of d columns that contains C as a subset and does not contain i (C' exists because $n \geq d+1$). Since A is the matrix of a d-CFF(t, n), there exists a row j such that $A_{j,i} = 1$ and $A_{j,c} = 0$ for all c in C'. Therefore $\mathsf{Batch}(B_j) = 1$ and σ_i is recognized as a valid signature in step 4 of the algorithm.

Remark 2.4. Shultz makes the following observation for batches containing $d' > d$ invalid signatures [33]. Let B' be the resulting set of signatures after removing all the signatures belonging to valid sub-batches, in step 4. If $|B'| > d$, the number of invalid signatures in the input batch exceeds d. In this case some valid signatures may be covered by $U(D)$, but are not present in a valid test. Thus B' contains all d' invalid signatures, but may contain some valid signatures as well.

A recent paper of Porat and Rothschild [31] explicitly constructs (n, d)-*strongly selective families* from error correcting codes. This structure is equivalent to a $(d-1)$-CFF(t, n) (see [12]), and hence it gives a nonadaptive ISF.

Theorem 2.5 ([31], Th. 1)**.** *It is possible to construct a* d-CFF(t, n) *with* $t = \Theta((d+1)^2 \log n)$ *in* $\Theta((d+1)n \log n)$ *time.*

In light of the bounds on t given in Appendix A, this construction is asymptotically optimal. We choose to ignore the constant hidden by the Θ-notation, as even with this assumption the CFF algorithm is outperformed by other methods.

2.3.2 Nonadaptive Group Testing with id-codes

The definition of identification codes is very general: any binary matrix which specifies a group testing algorithm is an id-code. Thus CFF are id-codes, and the d-separable property defined in 2.3.1 is both necessary and sufficient for an id-code. The construction of id-codes put forward in Pastuszak et al. [29] is a cover-free family with some additional constraints on the number of nonzero row and column entries. Using their construction gives the following ISF.

Theorem 2.6 ([29], Cor. 4)**.** *The number* t *of tests necessary to identify* d *bad signatures in a batch of size* n *satisfies* $t \leq (d+1)\sqrt{n}$*.*

Clearly, as $n \to \infty$ for fixed d, this method will require a much larger number of tests than CFF-based methods, since $\sqrt{n}$ dominates $\log n$. However, the CFF constructions presented have a quadratic dependence on d, while d is linear in Theorem 2.6. Therefore, for fixed n and increasing d, there will be a crossover point after which the id-code ISF outperforms the CFF ISF. Comparing the formulae,

$$\begin{aligned}(d+1)^2 \log(n) &< (d+1)\sqrt{n} \\ d &< \frac{\sqrt{n}}{\log n} - 1 \ .\end{aligned}$$

This gives the value of d in terms of n before which the CFF ISF outperforms the id-code ISF. For example, when $n = 10^3, 10^4, 10^5, 10^6$, d must be greater than $2, 6, 18, 49$ (resp.) for the id-code ISF to be more efficient.

2.3.3 Random Matrices

In this section we describe a probabilistic nonadaptive ISF which is based on a random matrix, and fails with a given probability. Du and Hwang give the probability that a random matrix is a d-CFF.

Theorem 2.7. *Let* C *be a random* $t \times n$ *binary matrix where* $C_{i,j} = 1$ *with probability* $q = 1/(d+1)$*. Then* C *is a* d-CFF(t, n) *with probability at least*

$$(d+1)\binom{n}{d+1}\left[1 - q(1-q)^d\right]^t \ .$$

Proof. Let D be a set of d columns of C, and let c a single column. In a single row, the probability that $c = 1$ and $D = 0, \ldots, 0$ is $q(1-q)^d$. (Note that $q = 1/(d+1)$ maximizes this probability.) The probability that this pattern does not occur in any of the t rows is $\left[1 - q(1-q)^d\right]^t$. Since the $d+1$ columns of D and c may be chosen in $(d+1)\binom{n}{d+1}$ ways, this gives the bound on the probability that C is a CFF stated in the theorem. □

Now we consider constructing an ISF as described at the beginning of Section 2.3.1 using random matrices. Certainly, this approach would succeed with probability at least that given by Theorem 2.7. However, the ISF will have significantly better performance, since the only case that affects our result is when the d columns corresponding to the bad signatures cover another column. If this occurs, then the covered column may be valid, but it will not appear in a valid test. Columns corresponding to valid signatures which cover each other will have no effect on the ISF. Therefore, we need only consider the probability that a *fixed set of d columns covers another column*. Since the d columns corresponding to defectives are fixed with respect to a batch, the remaining column may be chosen in $n-d$ ways, which gives the following result. The same improvement may be used in DNA library screening (see [16, Th. 9.3.3] and [2]).

Theorem 2.8. *There exists an ISF which identifies d defectives in a batch of size n using t tests with failure probability $P_{d,n} \leq (n-d)\left[1-q(1-q)^d\right]^t$, where $q = 1/(d+1)$.*

Remark 2.9. The error of this ISF is one-sided. It may output a valid signature as invalid. To detect this, we must individually test the output signatures, to confirm that they are invalid.

3 Comparison of Algorithms

In this section we compare the ISF algorithms given in Section 2. We compare them based on the number of tests, and their behaviour when d (the number of defectives) is unknown, or estimated incorrectly. Finally we discuss how the ISFs given by Law and Matt [24, 25] for a specific class of signature schemes compare to the generic ISF algorithms given in this paper.

3.1 Number of Tests

First, for each of the ISF algorithms in Section 2, we give the bound on the worst case number of calls to Batch (Table 1). Table 1 gives the bound for the trivial parallelization of (generalized) binary splitting: divide the original batch into p equal-sized sub-batches. The KUW algorithm is a better parallelization of binary splitting. For generalized binary splitting, the bounds given hold for $d \geq 2$, while for $d = 1$ the number of required tests is $\lfloor \log n \rfloor + 1$.

Next we compare the number of tests required by each method for various choices of n, d, and p (the number of processors available). In Ferrara et al. [17], the choices $n = 1024$, $d = 1, \ldots, 153$ were used when investigating the practical performance of the binary splitting method. In Pastuszak et al. [29], choices of $n \in [16, 1024]$ are used to give the average number of tests for the binary splitting method when $d = 1, \ldots, 16$. In Law and Matt [24], tables are given with $n = 2^4, 2^6, 2^8, 2^{10}, 2^{12}$ and $d = 1, \ldots, 4$. In Matt [25], the parameters chosen for comparison are $n = 2^4, 2^6, 2^8, 2^{10}$ and $d = 1, \ldots, n$ (here the goal was to show better performance with large d). All previous work considered $p = 1$, i.e., a single processor. We will compare the ISF algorithms with $n = 10^3, 10^4, 10^5, 10^6$, $d = 1, 2, 3, 4, 10$ and $p = 2, 4, 8, 16$. Table 2 gives the algorithm requiring the fewest tests when $p = 1$ and Table 3 provides the same information when $p \geq 2$ (according to the bounds in Table 1). A finer grained comparison is given in Appendix B, where Tables 5, 6 and 7 give the actual number of tests required under various combinations of parameters.

Discussion In the case of a single processor, (Tables 2 and 5), we find that the adaptive algorithms have the best performance. In particular, generalized binary splitting slightly outperforms

Method	Sec.	Tests (worst case)	Tests with p processors
Individual Testing	2.1	$n-1$	$\lceil n/p \rceil - p$
Binary Splitting (B.S.)	2.2.1	$d\lceil \log n \rceil$	$d\left\lceil \log\left(\frac{n}{p}\right)\right\rceil$
Gen. Bin. Splitting (G.B.S)	2.2.1	$d-1+\left\lceil \log\binom{n}{d}\right\rceil$	$d-1+\left\lceil \log\binom{n/p}{d}\right\rceil^{\dagger}$
Li's s-stage	2.2.2	$\frac{e}{\log e} d \log \frac{n}{d}$	$\frac{e}{\log e}\frac{d}{p}\log\frac{n}{dp} + \ln\frac{n}{dp} + d$
PR CFF	2.3.1	$(d+1)^2 \log n$	$((d+1)^2 \log n)/p$
PPS id-codes	2.3.2	$(d+1)\sqrt{n}$	$((d+1)\sqrt{n})/p$
KUW	2.2.1	$d\lceil \log_2 n \rceil$	$d\left\lceil \log_{p+1} n \right\rceil$

Table 1: Summary of the number of tests required for the ISF algorithms presented in §2. The number of tests required by the random matrices ISF must be computed using Theorem 2.8. "PR CFF" is the ISF based on Theorem 2.5, and "PPS id-codes" is the ISF in Theorem 2.6.

binary splitting, especially as d grows. With a single processor the KUW algorithm has the same performance as binary splitting, hence we have omitted it from the table.

When two or more processors are available to the ISF (Tables 3, 6 and 7), Li's s-stage algorithm and the KUW algorithm begin to show the best performance. The performance gap is most pronounced as the number of processors grows for any of the choices of (n, d) presented. In general, the nonadaptive algorithms improve when more processors are available, as they provide a speedup linear in the number of processors. Regarding the nonadaptive algorithms, the PR CFF algorithm (Th. 2.5) requires fewer tests than the PPS id-code algorithm (Th. 2.6) when $d < \sqrt{n}/\log n - 1$. If a failure probability of 0.001 is tolerable (see Remark 2.9), the random matrix ISF (RM ISF) outperforms the CFF and id-codes methods since it requires a weaker property from the matrix, as discussed following Theorem 2.7. The RM ISF with failure probability 0.001 is best overall when $p = 16$, $d = 4$ and $n = 10^4, 10^5, 10^6$ (see Appendix B). However, determining whether the RM ISF has failed requires d individual verifications.

In the detailed tables of Appendix B, there are many parameter combinations where multiple ISFs require a nearly equal number of tests. In these cases, implementation factors, average case performance, and the size of subset tests may influence the best choice.

3.2 Unknown Number of Invalid Signatures

Table 4 lists the behaviour of each of the algorithms when the true number of signatures, is d', a value different from our estimate d.

The binary splitting algorithm has a certain grace with respect to handling arbitrary d, in that the algorithm's behaviour is unchanged, and the bound on the number of tests holds as d changes. On the other hand, Li's s-stage algorithm, and generalized binary splitting begin by computing some parameters based on n and d in order to meet the performance bound stated in Table 1. If a batch contains $d' \neq d$ invalid signatures these parameters will not be chosen optimally, and it is

n	d	Fewest Tests
10^3	1–3	(generalized) binary splitting
	4,10	generalized binary splitting
10^4	1	(generalized) binary splitting
	2–4,10	generalized binary splitting
10^5	1,2	(generalized) binary splitting
	3,4,10	generalized binary splitting
10^6	1–3	(generalized) binary splitting
	4,10	generalized binary splitting

Table 2: Algorithm requiring the fewest number of tests when $p = 1$. The number of tests required by all algorithms listed in Table 1 is given in Table 5.

n	d	Fewest Tests when $p =$			
		2	4	8	16
10^3	4	KUW	LI	LI	LI
10^4	4	KUW	KUW	LI	LI
10^5	4	KUW	KUW	LI	LI
10^6	4	KUW	KUW	KUW	LI
10^3	10	LI	LI	LI	LI
10^4	10	KUW	LI	LI	LI
10^5	10	KUW	LI	LI	LI
10^6	10	KUW	LI	LI	LI

Table 3: Algorithm requiring the fewest number of tests with p processors. The number of tests required by all algorithms listed in Table 1 is given in Tables 6 and 7. Here, LI stands for Li's Algorithm (§2.2.2).

unclear to what extent this will hurt the performance of the algorithm. It is also unclear whether better performance is obtained by underestimating or overestimating d'.

When a batch contains $d' > d$ invalid signatures, the CFF and id-code algorithms output a set B' of ℓ signatures, where $d < \ell \leq n$. All d' defectives are in B'; however, it may contain valid signatures as well. As d' increases, ℓ will increase as well, and less information is gained. The case $d' > d$ is easily recognized (if $|B'| > d$), and we may restart the ISF with a larger estimate of d.

The random matrix ISF outputs each $d' > d$ with probability $P_{d',n}$, given in Theorem 2.8. For these algorithms we may run t tests to identify some valid signatures, remove them from the batch, re-estimate d, and re-run the ISF.

Another option when d is unknown is to use a *competitive algorithm*, i.e., one which assumes no *a priori* information about d, yet completes in a bounded number of tests (see [16, Ch. 4]). For example, the "jumping algorithm" of Bar-Noy et al. [3], identifies d invalid signatures in at most $1.65d(\log \frac{n}{d} + 1.031) + 6$ tests, for $0 \leq d \leq n$. Note that this flexibility comes at a cost because the performance of a competitive algorithm when d is known to be small is poorer than the other ISFs presented.

Algorithm	When $d' < d$	When $d' > d$
B.S.	Outputs d' invalid signatures in time $M_{\text{B.S.}}(d', n)$.	
G.B.S., Li	Outputs d' invalid signatures but using suboptimal parameter choices thus requiring extra work.	
KUW	Outputs d' invalid signatures in time $M_{\text{KUW}}(d', n)$.	
CFF, id-codes	returns d' invalid signatures	returns a set of $d \leq \ell \leq n$ potentially invalid signatures
RM	Outputs d' signatures in $M_{\text{RM}}(d, n)$ tests	Outputs d bad signatures with prob. $P_{d,n}$ and d' bad signatures with probability $P_{d',n}$ (see Th. 2.8)

Table 4: Behaviour of ISFs when the true number of invalid signatures d' differs from the estimated number d. Here, $M_A(d, n)$ represents the number of tests required by algorithm A for a batch of size n with d defectives.

3.3 Comparison to Non-Generic ISF Algorithms

Recall from Section 1.2 that a non-generic ISF is an ISF which is customized to a particular signature scheme, integrated into the Batch algorithm. In the single processor setting, the ISFs requiring the fewest number of tests were binary splitting and generalized binary splitting. Since the non-generic ISF given by Law and Matt [24, 25] outperforms binary splitting, their ISF will outperform the generic ISF algorithms presented here (for the pairing-based signature schemes to which it applies).

The faster choice in the parallel case would depend on how well the specialized ISFs described by Law and Matt parallelize. If their improved version of binary splitting yields an improved version of the KUW test (which is similar to binary splitting) then the parameter combinations where KUW is the best may be improved upon.

A general comparison is beyond the scope of this work since the units are different: number of calls to Batch() (this work) vs. number of multiplications in a finite field (Law and Matt).

4 Conclusion

We have introduced a number of new algorithms for finding invalid signatures in bad batches. For many parameter choices, and especially with multiple processors, the new methods outperform known methods. Our comparison shows that the best algorithm depends strongly on the choice of parameters, and no single algorithm is best in all cases. Topics for future work include: i) comparison of implementations to compensate for not considering the sizes of sub-batches, and ii) specializing the given ISFs to specific signature schemes, perhaps by using techniques from Law and Matt's specialized ISFs for pairing-based signature schemes.

Acknowledgements: Thanks are due to Urs Hengartner, Artur Jackson, and Aniket Kate for providing feedback on an earlier draft of this paper. This research was supported by an NSERC discovery grant, and PGS scholarship.

References

[1] M. Al-Ibrahim, H. Ghodosi and J. Pieprzyk. Authentication of concast communication. *Proc. of INDOCRYPT 2002, LNCS* **2551** (2002), 185–198.

[2] D.J. Balding, W.J. Bruno, E. Knill and D.C. Torney. A comparative survey of nonadaptive probing designs. *Genetic Mapping and DNA Sequencing, IMA Vol. in Math. and Its Applications* (1996), 133–154, Springer-Verlag.

[3] A. Bar-Noy, F.K. Hwang, I. Kessler and S. Kutten. Competitive group testing in high speed networks. *Discrete Applied Math.* **52** (1994), 29–38.

[4] M. Bellare, J. Garay and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. *Proceedings of EUROCRYPT 1998, LNCS* **1403** (1998), 236–250.

[5] D. Boneh, B. Lynn and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology* **17** (2004), 297319.

[6] C. Boyd and C. Pavlovski. Attacking and repairing batch verification schemes. *Proceedings of ASIACRYPT 2000, LNCS* **1976** (2000), 58–71.

[7] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. *Proceedings of SCN 2002*, LNCS **2576** (2002), 268–289.

[8] J. Camenisch, S. Hohenberger and M. Østergaard Pedersen. Batch verification of short signatures. *Proceedings of EUROCRYPT '07, LNCS* **4515** (2007), 246–263.

[9] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya and M. Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)* (2006), 201–210.

[10] J. Cha and J. Cheon. An identity-based signature scheme from gap Diffie-Hellman groups. *Proceedings of PKC 2003, LNCS* **2567** (2003), 18–30.

[11] S. Chatterjee and P. Sarkar. Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model. *Proceedings of the 8th International Conference on Information Security and Cryptology (ICISC), LNCS* **3935** (2005), 424–440.

[12] A.E.F. Clementi, A. Monti, and R. Silvestri. Distributed broadcast in radio networks of unknown topology. *Th. Comp. Sci.* **302** (2003), 337–364.

[13] A. De Bonis and U. Vaccaro. Constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels. *Th. Comp. Sci.* **306** (2003), 223–243.

[14] R. Dorfman. The detection of defective members of large populations. *Ann. Math. Statist.* **14** (1943), 436–440.

[15] D. Du and F.K. Hwang. *Combinatorial Group Testing and its Applications*. World Scientific, Singapore, 1993.

[16] D. Du and F.K. Hwang. *Combinatorial Group Testing and its Applications 2nd Edition*. World Scientific, Singapore, 2000.

[17] A.L. Ferrara, M. Green, S. Hohenberger and M. Østergaard Pedersen. Practical Short Signature Batch Verification *ePrint Report* **015/2008** (2008). Available online: `http://eprint.iacr.org/2008/015`

[18] A. Fiat. Batch RSA. *Proceedings of CRYPTO'89* (1989), 175–185.

[19] A. Fiat. Batch RSA. *Journal of Cryptology* **10** (1997), 75–88.

[20] J.A. Garay, J.N. Staddon and A. Wool. Long-lived broadcast encryption. *Proceedings of CRYPTO 2000, LNCS* **1880** (2000), 333–352.

[21] M.C. Hu, F.K. Hwang and J.K. Wang. A boundary problem for group testing. *SIAM J. Alg. Disc. Methods* **2** (1981), 81–87.

[22] R.M. Karp, E. Upfal and A. Wigderson. The complexity of parallel search. *J. Comput. Syst. Sci.* **36** (1988), 225–253.

[23] W.H. Kautz and R.G. Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory* **10** (1964), 363–373.

[24] L. Law and B.J. Matt Finding invalid signatures in pairing-based batches. *Proceedings of Cryptography and Coding 2007, LNCS* **4887** (2007), 34–53.

[25] B.J. Matt. Identification of multiple invalid signatures in pairing-based batched signatures. *Proceedings of PKC 2009, LNCS* **5443** (2009), 337–356.

[26] C.J. Mitchell and F.C. Piper. Key storage in secure networks. *Discrete applied mathematics* **21** (1988), 215–228.

[27] D. Naccache, D. M'raihi, S. Vaudenay and D. Raphaeli. Can DSA be improved? Complexity trade-offs with the digital signature standard. *Proceedings of EUROCRYPT '94, LNCS* **950** (1994), 77–85.

[28] Q.A. Nguyen and T. Zeisel. Bounds on constant weight binary superimposed codes. *Problems in Control and Information Theory* **17** (1988), 223–230.

[29] J. Pastuszak, D. Michalek, J. Pieprzyk and J. Seberry. Identification of bad signatures in batches. *Proceedings of PKC 2000, LNCS* **1751** (2000), 28–45.

[30] J. Pastuszak, J. Pieprzyk and J. Seberry. Codes identifying bad signatures in batches. *Proceedings of INDOCRYPT 2000, LNCS* **1977** (2000), 143–154.

[31] E. Porat and A. Rothschild. Explicit non-adaptive combinatorial group testing schemes. *Proceedings of Automata, Languages and Programming, ICALP 2008, LNCS* **5125** (2008), 748–759.

[32] M. Ruszinkó. On the upper bound of the size of the r-cover-free families. *Journal of Combinatorial Theory Series A* **66** (1994), 302–310.

[33] D.J. Shultz. Topics in nonadaptive group testing. Ph.D. Dissertation, Temple University, 1992.

[34] J. Spencer. Minimal completely separating systems. *Journal of Combinatorial Theory* **8** (1970), 446–447.

[35] E. Sperner. Ein Satz Uber Untermengen einer endliche Menge. *Math. Zeit.* **27** (1928) 544–548.

[36] J.N. Staddon, D.R. Stinson and R. Wei. Combinatorial properties of frameproof and traceability codes. *IEEE Trans. Inf. Theory* **47** (2001), 1042–1049.

[37] D.R. Stinson, T. van Trung and R. Wei. Secure frameproof codes, key distribution patterns, group testing algorithms and related structures. *Journal of Statistical Planning and Inference* **86** (2000), 595–617.

[38] D.R. Stinson, R. Wei and L. Zhu. Some new bounds for cover-free families. *Journal of Combinatorial Theory Series A.* **90** (2000), 224-234.

[39] D.R. Stinson. Attack on a concast signature scheme. *Information Processing Letters* **91** (2004), 39-41.

[40] M. Sobel and P.A. Groll. Group testing to eliminate efficiently all defectives in a binomial sample. *Bell System Tech. J.* **28** (1959), 1179–1252.

[41] J.H. van Lint. *Introduction to Coding Theory.* Springer, New York, 1998.

A Bounds and Constructions of Cover-Free Families

The number of rows, t, in the matrix representation of a d-CFF(t, n) gives the number of tests required in the group testing algorithm given in the previous section. In this section we present bounds for t since this indicates how well (at best) we can expect CFF-based nonadaptive group tests to perform.

First we present a necessary condition for the existence of CFF, a lower bound on the number of rows.

Theorem A.1 (see [38], Th. 1.1)**.** *For any $d \geq 1$, in a d-CFF(t, n)*

$$t \geq c \left(\frac{d^2}{\log d} \right) \log n$$

The constant c is approximately $1/8$ (shown in [32]).

It is immediately clear that the nonadaptive feature comes at a cost, since the number of tests will always be larger than $d \lceil \log(n) \rceil$, the number of tests required by binary splitting (c.f. 2.2.1).

De Bonis and Vaccaro bound t from the other direction.

Theorem A.2 ([13], Cor. 1)**.** *There exists a d-CFF(t, n) with*

$$t < 24d^2 \log(n + 2) \, .$$

Their proof method is constructive, based on a greedy algorithm, and it is efficient for small CFF.

A.1 Explicit Constructions of Cover-Free Families

The case $d = 1$. Construction of 1-CFF(t, n) is optimal and simple; choose all $\binom{t}{\lfloor t/2 \rfloor}$ length t binary vectors with weight $\lfloor t/2 \rfloor$. This is proven in Spencer [34], based on Sperner's theorem [35], and gives the following nonadaptive ISF.

Theorem A.3. *In a batch of n signatures, a single bad signature may be identified using t parallel tests, provided*

$$\binom{t}{\lfloor t/2 \rfloor} \geq n \; .$$

Therefore, 15, 20 and 25 tests may identify a single bad signature in a batch of size at most 6435, 184756 and 5200300, respectively.

CFF from codes with large distance. This construction is due to Kautz and Singleton [23] (and is it also described in [16]). They prove a lemma that gives conditions under which a constant weight binary code is a CFF. The codewords of a constant weight code all have a fixed number of nonzero coordinates. Now suppose we have an (N, n, q, ℓ) code, i.e. n codewords of length N over q symbols with minimum distance ℓ. We must replace the q-ary alphabet with a binary one. Let $\varphi : \{1, \ldots, q\} \to \{0, 1\}^{q_0}$ be a map which encodes $1, \ldots, q$ as binary vectors of length q_0. The codeword $(c_1, \ldots, c_N)$ will be replaced by $(\varphi(c_1), \ldots, \varphi(c_N))$.

A simple choice of φ encodes i as the i-th column of the $q \times q$ identity matrix. The resulting code has constant weight, and gives a d-CFF(qN, n) if $d < \frac{N-1}{N-\ell}$. This gives the following explicit construction.

Theorem A.4. *Let q be a prime power, $2 \leq k < q$ be an integer and C be a $(q, q^k, n, \ell = q-(k-1))$ Reed-Solomon code (see [41, §6.8]). Then C is d-CFF(q^2, n) if*

$$d < \frac{q-1}{k-1} \; .$$

For example, using Theorem A.4, we can construct an ISF for $n = 512$, $d = 3$ with $t = 64$ or for $n = 4096$, $d = 7$ with $t = 256$.

Alternate choices of φ are possible, so long as the encoding $\varphi(1), \ldots, \varphi(q)$ forms a d-CFF(q_0, q). Smaller values of q_0 yield CFF with fewer rows. We refer the reader to Kautz and Singleton [23], and Nguyen and Zeisel [28] for the details of such recursive constructions.

B Comparison Details

Table 5 gives the number of tests required by each algorithm when $p = 1$, with varying n and d, while Tables 6 and 7 fix $d = 4$ and $d = 10$ respectively, with varying n and p.

Method	$n = 10^3$, $d =$					$n = 10^4$, $d =$				
	1	2	3	4	10	1	2	3	4	10
Binary Splitting	10	20	30	40	100	14	28	42	56	140
Gen. Bin. Splitting	10	20	30	39	87	14	27	40	52	121
Li's s-stage	18	33	47	60	125	25	46	66	85	187
PR CFF	13	89	159	249	1205	16	119	212	332	1607
PPS id-codes	63	94	126	158	347	200	300	400	500	1100
Random Matrices	49	87	124	162	387	57	101	145	189	452
	$n = 10^5$					$n = 10^6$				
Binary Splitting	17	34	51	68	170	20	40	60	80	200
Gen. Bin. Splitting	17	34	50	65	154	20	40	60	79	187
Li's s-stage	31	58	84	110	250	37	71	103	135	312
PR CFF	20	149	265	415	2009	23	179	318	498	2411
PPS id-codes	632	948	1264	1581	3478	2K	3K	4K	5K	11K
Random Matrices	65	115	166	216	517	73	130	186	243	581

Table 5: Table showing the number of tests required by each group testing algorithm from Table 1 when $n = 10^3, 10^4, 10^5, 10^6$ and $d = 1, 2, 3, 4, 10$. For random matrices a success probability of 99.9% is required.

Method	$d = 4$							
	$n = 10^3$, $p =$				$n = 10^4$, $p =$			
	2	4	8	16	2	4	8	16
Binary Splitting	36	32	28	24	52	48	44	40
Gen. Bin. Splitting	35	31	27	23	48	44	40	36
KUW	28	20	16	12	36	24	20	16
Li's s-stage	35	19	12	8	49	27	17	12
PR CFF	125	63	32	16	166	83	42	21
PPS id-codes	79	40	20	10	250	125	63	32
Random Matrices	81	41	21	11	95	48	24	12
	$n = 10^5$				$n = 10^6$			
Binary Splitting	64	60	56	52	76	72	68	64
Gen. Bin. Splitting	61	57	53	49	75	71	67	63
KUW	44	32	24	20	52	36	28	20
Li's s-stage	64	36	22	16	79	45	28	20
PR CFF	208	104	52	26	249	125	63	32
PPS id-codes	719	396	198	99	2.5K	1250	625	313
Random Matrices	108	54	27	14	122	61	31	16

Table 6: Table showing the number of tests required by each group testing algorithm from Table 1 when $n = 10^3, 10^4$, $d = 4$ and the number of processors available is $p = 2, 4, 8, 16$. For random matrices a success probability of 99.9% is required.

Method	$d = 10$							
	$n = 10^3$, $p =$				$n = 10^4$, $p =$			
	2	4	8	16	2	4	8	16
Binary Splitting	90	80	70	60	130	120	110	100
Gen. Bin. Splitting	77	67	57	47	111	101	91	80
KUW	70	50	40	30	90	60	50	40
Li's s-stage	67	35	21	14	100	53	31	21
PR CFF	603	302	151	76	804	402	201	101
PPS id-codes	174	87	44	22	550	275	138	69
Random Matrices	194	97	49	25	226	113	57	29
	$n = 10^5$, $p =$				$n = 10^6$, $p =$			
	2	4	8	16	2	4	8	16
Binary Splitting	160	150	140	130	190	180	170	160
Gen. Bin. Splitting	144	134	124	114	177	167	157	147
KUW	110	80	60	50	130	90	70	50
Li's s-stage	134	70	41	27	167	88	51	33
PR CFF	1005	503	252	126	1206	603	302	151
PPS id-codes	1739	870	435	218	5500	2750	1375	688
Random Matrices	259	130	65	33	291	146	73	37

Table 7: Table showing the number of tests required by each group testing algorithm from Table 1 when $n = 10^3, 10^4$, $d = 10$ and the number of processors available is $p = 2, 4, 8, 16$. For random matrices a success probability of 99.9% is required.