Identification of Bad Signatures in Batches

Jarosław Pastuszak[1], Dariusz Michałek[1], Josef Pieprzyk[2], and Jennifer Seberry[2]

[1] Systems Research Institute
Polish Academy of Sciences
Warsaw, POLAND
jarek.pastuszak@bsb.com.pl

[2] Centre for Computer Security Research
School of IT and Computer Science
University of Wollongong
Wollongong, NSW 2522, AUSTRALIA
Josef_Pieprzyk@uow.edu.au
Jennifer_Seberry@uow.edu.au

Abstract. The paper addresses the problem of bad signature identification in batch verification of digital signatures. The number of generic tests necessary to identify all bad signatures in a batch instance, is used to measure the efficiency of verifiers. The divide-and-conquer verifier $DCV_\alpha(x, n)$ is defined. The verifier identifies all bad signatures in a batch instance x of the length n by repeatedly splitting the input into α sub-instances. Its properties are investigated. In particular, probability distributions for the number of generic tests necessary to identify one, two and three bad signatures, are derived. The average numbers of GT tests necessary to identify bad signatures ranging from 1 to 16 are obtained from computer simulation. Further, a Hamming verifier (HV) is defined which allows to identify a single bad signature in a batch of the length $n = 2^k - 1$ using $k + 2$ tests. HV is generalised into the two-layer Hamming verifier (2HV). Given a batch instance of the length $2^k - 2$, the 2HV verifier identifies a single bad signature using $k + 2$ tests and two bad signatures in expense of $3k + 3$ tests. The work is concluded by comments about a general model for verification codes identifying t bad signatures and the design of verifiers using combinatorial structures.

1 Introduction

Digital signatures are main cryptographic tools for message authentication. Unlike hand-written signatures, digital ones differ from one document to another as they produce a fingerprint which reflects both the identity of signer (or more precisely their secret signing key) and the contents of the document (typically embedded in its digest). Any digital signature includes signing and verification algorithms. The signing algorithm can be run by the holder of the secret signing key. The verification algorithm can be run by everybody as the matching (verification) key is public.

Often a signature is generated once but its verification is done many times. A growing usage of digital signatures for electronic payment systems stresses the need for streamlining of the signature verification. Batch verification offers an efficient verification of a collection of related signatures at a cost of making a mistake. The probability of mistake can be traded off with the efficiency. Batch verification is an option if the signature used exhibits the homomorphic property.

The idea of batch verification was spelt out in many papers [2,4,6,8].

2 Motivation

Undoubtedly, fast signature verification seems to be of utmost importance when there is a need for continual processing of many signatures. As shown by Bellare, Garay and Rabin in [1] there are three generic test which can be used for fast batch verification of signatures. Efficiency of these tests varies and depends on the size of a signature batch being verified. The main problem with batch verification is that they trade efficiency with security. In the case of individual signature verification, an attacker is forced to break the underlying signature scheme if they want to generate a valid signature for a message. In the case when the batch verification is applied, the attacker may also explore weaknesses existing in the verification tests. Verification tests are probabilistic algorithms for which it is possible to set the bound on the probability of acceptance of invalid signatures in the batch tested. As there is a direct relation between the probability and efficiency, one can expect that the probability may be lowered during the time when the heavy processing is expected (typically, the end of the week). Instead of breaking the underlying signature, attackers are encouraged to generate messages with invalid signatures on a massive scale. This serves two purposes. The first purpose is to increase the verification load, and one can expect that the manager responsible for verification of signatures, will lower the threshold probability even further. The second purpose is to increase the probability of attacker success. On the top of this, the attacker may have specific knowledge about which test will be used and what parameters are employed. This knowledge may give some hints as to how invalid signatures could be produced to maximise the chance of slipping through the tests.

When a collection of signatures passes the tests, the verifier accepts all the signatures as valid. Otherwise, the collection is rejected. Now the verifier must separate the valid signatures from invalid ones. In this paper, we consider different methods of invalid signature identification and evaluate efficiency of tests with invalid signature identification.

3 Background

There are two homomorphic operations widely used for signing: modular exponentiation (the base is fixed) and RSA exponentiation (the exponent is fixed). Consider modular exponentiation defined for a cyclic group of order q, where g is the cyclic group generator. The DSA or DSS signatures and their versions are signatures of this kind. Being more precise, the exponents are computed individually for each signature. This computation is cheap – it takes one modular inversion and multiplication. The final verification can be done in batches in which exponents are added (see [5]).

Given a batch $x = ((m_1, s_1), \ldots, (m_n, s_n))$ of messages with their signatures, signatures can be verified one by one by checking

$$g^{m_i} \stackrel{?}{=} s_i \text{ for } i = 1, \ldots, n$$

The cost of verification is n exponentiations. To reduce the number of expensive exponentiations and speed up the verification process, one can verify the following

$$V_g(x) \equiv \left(g^{\sum_{i=1}^{n} m_i} \stackrel{?}{=} \prod_{i=1}^{n} s_i \right) \tag{1}$$

This costs one exponentiation, $n-1$ modular multiplications, and $n-1$ modular additions. Typically, the calculation of $\sum_{i=1}^{n} m_i$ is done modulo q while $\prod_{i=1}^{n} s_i$ is performed modulo p where q divides $p-1$.

Consider the RSA exponentiation where the modulus N is the product of two primes p and q. The signer secret key is d and the public verification key is e. All signed messages are smaller than the modulus N. A typical batch of signatures looks like $((m_1, s_1), \ldots, (m_n, s_n))$. Sequential verification of the batch

$$s_i^e \stackrel{?}{=} m_i \text{ for } i = 1, \ldots, n,$$

takes n exponentiations. Again, the verification process can be sped up by using

$$V_e(x) \equiv \left(\prod_{i=1}^{n} m_i \stackrel{?}{=} \left(\prod_{i=1}^{n} s_i \right)^e \right) \tag{2}$$

This takes one exponentiation and $2(n-1)$ modular multiplications.

In general, a batch verifier is a probabilistic algorithm B which takes a batch instance $x = ((m_1, s_1), \ldots, (m_n, s_n))$ and a security parameter ℓ. The algorithm

– outputs "0" always whenever all the signatures in the batch are correct,
– outputs "1" with probability $1 - 2^{-\ell}$ whenever the batch contains incorrect signatures.

A batch verifier never makes mistakes when the batch is "clean". If the batch is "dirty" or contains incorrect signatures, then the batch verifier makes mistakes with probability $2^{-\ell}$.

There is a universal test which is applicable for any signature scheme which has a homomorphic property. The test (in [1] called random subset test) is defined as follows.

Definition 1. *Given a batch instance $x = ((m_1, s_1), \ldots, (m_n, s_n))$ and a security parameter ℓ. The universal test (UT) takes ℓ rounds. For each round*

1. *pick a random set $\mathcal{T} = \{t_1, \dots, t_n\}$, i.e. each t_i is selected independently and with the same probability from $\{0, 1\}$,*
2. *create a subset $x_{\mathcal{T}} = \{(m_i, s_i) | t_i = 1\}$,*
3. *run the test $V(x_{\mathcal{T}})$ (either $V_g(x_{\mathcal{T}})$ or $V_e(x_{\mathcal{T}})$). If the test accepts go to the next round. Otherwise, reject the batch.*

A useful test for signatures based on a fixed base applies a random string of small integers used in the test as exponents (in [8] called small exponents test).

Definition 2. *Given a batch instance $x = ((m_1, s_1), \dots, (m_n, s_n))$ and a security parameter ℓ. The small exponent (SE) test:*

1. *select at random a collection of small integers $e = (e_1, \dots, e_n)$ where $e_i < 2^\ell$,*
2. *convert the instance x into $x_e = ((m_1 e_1, s_1^{e_1}), \dots, (m_n e_n, s_n^{e_n}))$,*
3. *run the test $V_g(x')$. If the batch instance x' passes the test accept x otherwise reject.*

Clearly, we are interested in a generic test which always succeed when all signatures are valid and fails with an overwhelming probability when there is one or more bad signatures.

Definition 3. *Given a batch instance $x = ((m_1, s_1), \dots, (m_n, s_n))$. The generic test (GT) takes a batch instance x and*

1. *outputs "0" whenever all signatures are valid. The test never makes mistakes for this case,*
2. *returns "1" whenever there is at least one bad signature. In this case the test makes mistakes with probability $2^{-\ell}$.*

If a batch of signatures passes tests, then the verifier accepts the whole batch. The probability of mistake can be make small enough say smaller than 2^{-100}. However when a batch fails a test, the verifier is not able to reject all signatures in the batch. The verifier faces the problem of identification of bad signatures. Let us consider some possible solutions for bad signature identification.

The simplest solution for it could be based on testing all signatures one by one using the GT test.

Definition 4. Naive Verifier. *Given a batch instance $x=((m_1,s_1),...,(m_n,s_n))$.*

1. *Run GT(x, n). If GT(x, n)=0, accept the instance x and exit. Otherwise, when GT(x, n)=1, for $i = 1$ to $i = n$ do:*
   - *apply GT(x_i, 1),*
   - *if GT(x_i, 1) = 1 then store x_i otherwise go for the next i.*
2. *Output all stored signatures in the list NV(x).*

where $x_i = (m_i, s_i)$.

The well-known twelve-coin problem is very much related to the identification of bad signatures. It can be formulated as follows.

> Given 12 coins all of equal weight, except one defective coin. It is not known whether the defective coin is lighter or heavier than each of the others. Assume that there is a set of two-dish scales which can be used to carry out tests. Coins can be placed on both sides and if the weights are equal then the scales balance, otherwise they tilt downwards on the side carrying the heavier weight.
> Find a sequence of tests which can be performed using the scales to identify the defective coin within the three weightings only.

Note that identification of a bad signature resembles the twelve-coin problem. The main difference is that tests performed on batches do not allow us to see how the scales tilt. In other words, the tests carried out on batches allow us to see whether the batch is clean (the scales balance) or dirty (the scales do not balance).

4 Divide-and-Conquer Verifiers

Identification of bad signatures can be implemented by the so called divide-and-conquer (DC) verifier. The idea seems to be straightforward and can be traced in the literature under the name "cut and choose" [4].

The verifier is an algorithm which takes a batch instance x and outputs either "0" when the batch instance is clean otherwise returns a list of all bad signatures. It is defined as a recursive function.

Definition 5. DC Verifier. *Given a batch instance* $x = ((m_1, s_1), \ldots, (m_n, s_n))$ *with* $n = 2^k$ *signatures.*

1. Stopping case: *If the instance consists of* $n = 1$ *signature, then run the generic test on the input, i.e. GT(*$x, 1$*).If* $GT(x, 1) = 0$*, return* 0 *and exit. Otherwise output the bad signature and exit.*
2. *If the instance consists of* $n \neq 1$ *signature, apply the generic test on the input sample, i.e. GT(*x, n*). If* $GT(x, n) = 0$*, return* 0 *and exit. Otherwise go to the recursive step.*
3. Recursive step: *Divide the instance* x *into* α *batch instances* $(x_1, \ldots, x_\alpha)$ *containing* $\frac{n}{\alpha}$ *signatures each. The division is done at random. Call the DC verifier for* α *sub-instances, i.e.* $DCV_\alpha(x_1, \frac{n}{\alpha}) \cdots DCV_\alpha(x_\alpha, \frac{n}{\alpha})$*.*

The computational overhead of our verifiers is measured by the number of times the GT test is called during verification process. The worst case occurs when a batch instance contains all bad signatures. So the maximum number of tests performed by DCV$_\alpha$ is

$$\#\max(DCV_\alpha, n) = \sum_{i=0}^{k} \alpha^i = \frac{\alpha^{(k+1)} - 1}{\alpha - 1} = \frac{n\alpha - 1}{\alpha - 1} \qquad (3)$$

where α indicates that the DCV verifier slices input instances into α sub-instances of the same length and n is the length of the input batch instance.

From Equation (3), it is easy to observe that for very badly contaminated instances, the selection of a large α is preferred. Note that if $\alpha = n = 2^k$, then the DCV verifier becomes the NV verifier which always consumes $n+1$ tests.

4.1 Degree of Contamination Versus Parameter α

It is an interesting to ask about the degree of contamination of batch instances for which the naive verifier becomes more efficient than DCV$_2$. This is an important issue for efficient signature verification. To answer this question, assume that a batch instance consists of $n = 2^k$ signatures contaminated with $t = 2^r$ bad signatures ($r < k$). Denote the maximum numbers of GT tests necessary to identify all t bad signatures out of total n ones using the NV and DCV$_2$ verifiers by $\#\max(NV, n, t)$ and $\#\max(DCV_2, n, t)$, respectively.

Note that the worst case occurs when the DC verifier after the r-th recursive step all sub-instances contain precisely one bad signature. To get to this point, DCV$_2$ consumes precisely $2^r - 1$ tests. So

$$\#\max(DCV_2, 2^k, t) = 2^r - 1 + 2^r \times \#\max(DCV_2, 2^{k-r}, 1).$$

A single bad signature in a batch instance of size 2^{k-r} is always identifiable using $2(k-r)+1$ tests. Therefore, we obtain

$$\#\max(DCV_2, 2^k, t) = 2^{r+1}(k - r + 1) - 1.$$

Now we can ask how small the contamination of a batch instance should be to render the DCV verifier more efficient or

$$\#\max(DCV_2, 2^k, t) < \#\max(NV, 2^k, t).$$

If we substitute values obtained, then the inequality becomes

$$2^{(r+1)}(k - r + 1) - 1 < 2^k + 1$$

or equivalently

$$k - r + 1 < 2^{k-r-1} + 2^{-r}.$$

It is easy to check that this inequality holds for any $k - r \geq 3$. So we have proved the corollary.

Corollary 1. *DCV$_2$ is more efficient (consumes less GT tests) from the NV verifier if batch instances of 2^k signatures contain less than 2^{k-3} bad ones.*

Note that we have compared DCV$_2$ (binary split of batch instances) with DCV$_n$ (equivalent to NV). Similar considerations can be made for any two verifiers DCV$_\alpha$, DCV$_\beta$ for $\alpha \neq \beta$. This makes sense if the contamination varies and the parameter α can be adjusted accordingly.

Results of computer simulation conducted to determine the relation between the degree of contamination and the parameter α are summarised in Table 1.

Table 1. Tradeoff between parameter α and the degree of batch contamination

Number n	Number of bad signatures t	Optimal Parameter α
128	1	2, 4
	$2, 4$	4, 8
	8	32
	16	32, 64
	32	128
256	1	2, 4
	$2, 4, 8$	4
	16	8
	$32, 64$	64
512	1	2, 4
	$2, 4, 8, 16$	4
	$32, 64$	128
1024	$1, 2$	2, 4
	$4, 8$	4
	$16, 32$	4, 8
	64	256
	128	512
2048	1	2, 4
	$2, 4, 8, 16, 32, 64$	4
	$128, 256$	512
4096	1	2,4
	$2, 4, 8, 16, 32, 64, 128$	4
	$256, 512$	1024

4.2 Number of Tests Needed to Identify t Bad Signatures

Denote $\#(DCV_\alpha, n, t)$ to be the number of GT tests necessary to identify bad signatures from a batch instance with n signatures provided t ones are bad. As the DCV$_\alpha$ verifier is probabilistic in its nature, the number $\#(DCV_\alpha, n, t)$ is in fact a random variable. To simplify our notation, let

$$N_\alpha(t, n) = \#(DCV_\alpha, n, t).$$

Our aim is to derive the probability distribution for the variable $N_2(t, n)$.

Consider the verifier DCV$_2$ and the corresponding random variable $N_2(t, n)$. Let $t = 1$. Obviously, the verifier needs to perform $2k + 1$ tests, i.e.

$$N_2(1, 2^k) = 2k + 1.$$

This number of tests is constant and occurs with probability 1. By the way, the number of tests can be cut almost by half if $t = 1$ is known before hand as $N_2(1, 2^k) = k + 1$. This observation of course may be used for optimisation of the DCV verifier. This is especially effective for $\alpha = 2$. If a sub-instance passes the GT test, the second sub-instance is not tested (as it must fail it anyway). Instead, it is divided into halves and one of the resulting sub-instances is tested.

Let $t = 2$. Note that random variable $N_2(2, 2^k)$ can be expressed by random variables $N_2(2, 2^{k-1})$ and $N_2(1, 2^{k-1})$ according to the following equation:

$$N_2(2, 2^k) = \begin{cases} 1 + 2N_2(1, 2^{k-1}) \text{ with probability } p_{1,0} \\ 2 + N_2(2, 2^{k-1}) \text{ with probability } p_{2,0} \end{cases} \tag{4}$$

Similarly, we can write

$$N_2(2, 2^{k-1}) = \begin{cases} 1 + 2N_2(1, 2^{k-2}) \text{ with probability } p_{1,1} \\ 2 + N_2(2, 2^{k-2}) \text{ with probability } p_{2,1} \end{cases} \tag{5}$$

For $i = 2, \ldots, k-1$, we can generalise as

$$N_2(2, 2^{k-i}) = \begin{cases} 1 + 2N_2(1, 2^{k-i-1}) \text{ with probability } p_{1,i} \\ 2 + N_2(2, 2^{k-i-1}) \text{ with probability } p_{2,i} \end{cases} \tag{6}$$

Assume that at step j, two bad signatures clustered together in a single instance have been put into two different sub-instances. This means that the bad signatures were placed in the same instance j times in a row. Therefore

$$N_2(2, 2^k)(j) = 2j + 1 + 2N_2(1, 2^{k-j-1}) = 4k - 2j - 1 \tag{7}$$

where $j = 0, 1, \ldots, k-1$.

Now we are ready to calculate probabilities $p_{i,j}$. The parameter $n = 2^k$. The probability $p_{1,0}$ expresses the probability that the initial batch instance splits into two sub-instances containing one bad signature each so

$$p_{1,0} = \frac{\binom{2}{1}\binom{n-2}{\frac{n}{2}-1}}{\binom{n}{\frac{n}{2}}} = \frac{n}{2(n-1)}.$$

Similarly, the probability that after the split, one of the sub-instances contains two bad signatures is:

$$p_{2,0} = 2 \times \frac{\binom{2}{0}\binom{n-2}{\frac{n}{2}}}{\binom{n}{\frac{n}{2}}} = \frac{n-2}{2(n-1)}.$$

The multiplier 2 indicates the fact that two bad signatures can be in the first or the second sub-instance. Continuing our calculations, we obtain

$$p_{1,i} = \frac{n}{2(n-2^i)} \tag{8}$$

$$p_{2,i} = \frac{n-2^{i+1}}{2(n-2^i)} \tag{9}$$

The probability $p(j)$ that for some step j, two bad signatures have been placed into two different sub-instances is:

$$\begin{aligned} p(0) &= p_{1,0} \\ p(1) &= p_{2,0} \times p_{1,1} \\ &\vdots \\ p(j) &= p_{2,0} \times p_{2,1} \times \ldots \times p_{2,j-1} \times p_{1,j} \end{aligned}$$

After substituting values, the above equation takes on the following form:

$$p(j) = \frac{n}{n-1}\frac{1}{2^{j+1}}$$

for $j = 0, \ldots, k-1$ and $n = 2^k$. So we have proved the following corollary.

Corollary 2. *Given the DCV verifier with $\alpha = 2$. If a batch instance of length $n = 2^k$ is contaminated by two bad signatures, then the number $N_2(2, n)$ of necessary GT tests is a random variable whose probability distribution is as follows:*

$$P(N_2(2, n) = 4k - 2j - 1) = \frac{n}{n-1}\frac{1}{2^{j+1}} \tag{10}$$

for $j = 0, 1, \ldots, k-1$.

Now we derive the probability distribution for the required number of GT tests when the input batch instance is contaminated by three bad signatures ($t = 3$).

The number of GT tests is denoted by $N_2(3, 2^k)$. The number of tests satisfies the equation

$$N_2(3, 2^k) = \begin{cases} 1 + N_2(1, 2^{k-1}) + N_2(2, 2^{k-1}) & \text{with probability } p_1 \\ 2 + N_2(3, 2^{k-1}) & \text{with probability } p_2 \end{cases}$$

It means that after the first step, the verifier may split the input instance into two sub-instances where (1) one sub-instance contains one bad signature and the other sub-instance is contaminated by two bad signatures, (2) one sub-instance is clean and the other includes 3 bad signatures. The probability p_1 is equal to

$$p_1 = \frac{\binom{3}{1}\binom{n-3}{\frac{n}{2}-1}}{\binom{n}{\frac{n}{2}}} = \frac{3n}{4(n-1)}.$$

and the probability p_2 is

$$p_2 = 2 \times \frac{\binom{3}{0}\binom{n-3}{\frac{n}{2}}}{\binom{n}{\frac{n}{2}}} = \frac{n-4}{4(n-1)}.$$

Assuming that the bad signatures have been tossed into two sub-instances at the first step by the verifier, then the probability distribution can be derived from previous considerations (see Equation 8) and

$$P(N_2(3,n) = 6k - 2j - 5|(1,2)) = p_1 \times p_{1,1} = \frac{3n}{4(n-1)} \frac{n}{(n-2)} \frac{1}{2^{j+1}}$$

for $j = 0, 1, \ldots, k-2$.

Consider the case when bad signatures have been tossed into the same sub-instance (the other sub-instance is clean) – the case (0,3). Assume that for certain step i, the three bad signatures have been split into either (1,2) or (2,1). It means also that three bad signatures were tossed together i times so

$$N_2(3,n) = 2i + 1 + N_2(1, 2^{k-i-1}) + N_2(2, 2^{k-r-1})$$

for $i = 0, \ldots, k-1$. After substituting the expressions obtained for $t = 2$ and $t = 1$, we obtain final probability distribution.

Corollary 3. *Given the verifier DCV_α with $\alpha = 2$. If a batch instance of length $n = 2^k$ is contaminated by three bad signatures, then the number $N_2(3,n)$ of required GT tests is a random variable whose probability distribution is as follows:*

$$P(N_2(3,n) = 6k - 4i - 2j - 5) = \frac{3n^2}{(n-1)(n-2)} \frac{1}{2^{2i+j+3}} \frac{n - 2^{i+1}}{n - 2^{k-i-1}} \tag{11}$$

for $i = 0, 1, \ldots, k-1$ and $j = 0, 1, \ldots, k-i-2$.

Knowing the probability distributions for the number of GT tests necessary to identify bad signatures in the cases when $t = 1, 2, 3$, it is easy to find the average number of test. For the number of bad signatures $t > 3$, the average can be estimated using computer simulation. The results are compiled in Table 2.

4.3 Optimisation of DC Verifiers

As observed above, for DCV$_2$, the number of GT tests can be reduced if the verifier knows the precise number of bad signatures. If there is only a single bad signature ($t = 1$), then at each step the DCV$_2$ verifier needs to tests only single sub-instance out of two generated from the contaminated instance. If the sub-instance is clean, then the other sub-instance is dirty (and vice versa). So the number $N_2(1, 2^k) = 2k + 1$ can be reduced to $k + 1$. Even if the number of bad signatures is not known before hand, this observation can be exploited to reduce the number of GT tests.

Definition 6. Fast DC Verifier. *Given a batch instance $x = ((m_1, s_1), \ldots, (m_n, s_n))$ with $n = 2^k$ signatures.*

1. Stopping case: *If the instance consists of $n = 1$ signature, then run the generic test on the input, i.e. $GT(x,1)$.If $GT(x,1) = 0$, exit. Otherwise output the bad signature and exit.*

Table 2. The average number of GT tests necessary to identify t bad signatures in a sequence of length n

t	n=16	n=32	n=64	n=128	n=256	n=512	n=1024
0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
1	9,0	11,0	13,0	15,0	17,0	19,0	21,0
2	13,5	17,3	21,2	25,1	29,1	33,0	37,0
3	17,1	22,5	28,2	34,0	39,8	45,8	51,7
4	19,9	26,9	34,3	41,9	49,7	57,6	65,5
5	22,2	30,8	39,8	49,3	58,9	68,7	78,5
6	24,1	34,2	44,9	56,0	67,5	79,2	91,0
7	25,6	37,2	49,5	62,4	75,7	89,3	103,0
8	27,0	39,9	53,8	68,4	83,5	99,0	114,7
9	28,0	42,4	57,8	74,1	91,0	108,3	125,9
10	28,9	44,6	61,5	79,5	98,2	117,4	136,9
11	29,6	46,6	65,1	84,7	105,1	126,1	147,6
12	30,2	48,5	68,4	89,6	111,8	134,7	158,0
13	30,6	50,2	71,5	94,3	118,3	143,0	168,2
14	30,9	51,7	74,5	98,9	124,5	151,1	178,2
15	31,0	53,1	77,3	103,3	130,6	159,0	188,0
16	31,0	54,4	80,0	107,5	136,6	166,7	197,6

2. *If the instance consists of $n \neq 1$ signature, apply the generic test on the input sample, i.e. $GT(x, n)$. If $GT(x, n) = 0$, exit. Otherwise go to the recursive step.*
3. Recursive step: *Split the instance x into α batch instances $(x_1, \ldots, x_\alpha)$ containing $\frac{n}{\alpha}$ signatures each. The split is done at random. Call the DC verifier for $\alpha - 1$ sub-instances, i.e. $DCV(x_1, \frac{n}{\alpha}) \cdots DCV(x_{\alpha-1}, \frac{n}{\alpha})$. If there is at least one dirty sub-instance, call $DCV(x_\alpha, \frac{n}{\alpha})$. Otherwise (i.e. if all sub-instances are clean), call the verifier $DCV(x_\alpha, \frac{n}{\alpha})$ in which the GT test is skipped.*

Note that the fast verifier DCV_2 needs $\approx (1.5k + 1)$ tests (instead of $2k + 1$) if there is one bad signature (but the verifier does not know this before hand). The advantage drops if α grows. In general, the fast verifier DCV_α consumes $((\alpha - 1 + \frac{1}{\alpha})k + 1)$ tests instead of $(\alpha k + 1)$ assuming a single bad signature and the length of batch instance α^k.

Further improvement can be achieved if the split of instances is not random. It turns out that if the random split into sub-instances is replaced by deterministic split into α sub-instances, then the number $N_\alpha(t, n)$ preserve the same probability distribution assuming that the input batch instance is random. This assumption seems to hold in most practical situations.

Additionally, the DCV verifier can be sped up by a careful design of the GT test. To illustrate the point assume that the DCV verifier is used to identify bad signatures by running the test $V_e(x)$ defined by Equation (2). Given a batch instance $x = ((m_1, s_1), \ldots, (m_n, s_n))$. Note that the test $V_e(x)$ is run for the whole instance x and needs to produce the product of all messages ($\prod_{i=1}^{n} m_i$) and all signatures ($\prod_{i=1}^{n} s_i$). Before calling the verifier, we can create two multiplication tables for messages and signatures. For instance, the message multiplication table is of the following form (the input instance is of length 16):

$$\begin{array}{rc} \text{Batch Instance:} & 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 \\ \text{1-st level of products:} & (1,2)(3,4)(5,6)(7,8)(9,10)(11,12)(13,14)(15,16) \\ \text{2-nd level of products:} & (1,2,3,4)(5,6,7,8)(9,10,11,12)(13,14,15,16) \\ \text{3-rd level of products:} & (1,2,3,4,5,6,7,8)(9,10,11,12,13,14,15,16) \\ \text{4-th level of products:} & (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16). \end{array}$$

where (i,j) stands for the product of $m_i \times m_j$. All multiplications needed by the DCV verifier are already stored in the tables. To run the test $V_e(x)$, it needs to perform a single exponentiation.

5 Verifiers Based on Hamming Codes

Assume that batch instances are contaminated by at most a single bad signature. This assumption is true most of the time when the source of errors is unreliable storage or communication so from time to time some signatures (or corresponding messages) get corrupted. Given a batch instance $x = ((m_1, s_1), \ldots, (m_n, s_n))$ of length $n = 2^k - 1$ for some positive k. To identify a single bad signature, it is enough to design a Hamming code with the block length n and k parity check equations. Let H be a parity check matrix. H contains k rows and n columns. If the matrix H has the form

$$H = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \end{bmatrix} = \begin{bmatrix} 1 \; 2 \; 3 \ldots 2^k - 1 \end{bmatrix}$$

where $h_i = (h_{i,1}, \ldots, h_{i,n})$ is a binary string of length n with the weight 2^{k-1} and integers i in the matrix represent columns which are binary strings representing the integer. Note that the Hamming code with such H allows for a quick identification of error position as the error syndrome is the binary index of the position in which the error occurs (for details see [3]).

Definition 7. Hamming Verifier. *Given a batch instance $x = ((m_1, s_1), \ldots, (m_n, s_n))$ of length $n = 2^k - 1$ for some positive k.*

1. *Apply the generic test on the input instance. If $GT(x,n)$=0, exit. Otherwise, go to the next step.*
2. *Create k sub-instances. i.e.*

$$x_i = \{(m_j, s_j) | h_{i,j} = 1\}$$

for $i = 1, \ldots, k$ where x_i is a sub-instance composed from elements of x chosen whenever $h_{i,j}$ is equal to 1 (elements of x for which $h_{i,j}$=0 are ignored).

3. *Run GT($x_i, 2^{k-1}$) $= \sigma_i$ for $i = 1, \ldots, k$ where $\sigma_i = 0$ if the test accepts x_i or $\sigma_i = 1$ if it fails. The syndrome $(\sigma_1, \ldots, \sigma_k)$ identifies the position of the bad signature.*
4. *Apply the generic test on the input instance without the bad signature. If the batch instance is accepted, return the index of the bad signature. Otherwise, the verifier fails and exits* 1.

The Hamming verifier (HV) succeeds whenever batch instances of the length $2^k - 1$ are contaminated by single bad signatures and HV consumes $k + 2$ GT tests. This number is almost identical to the number which is needed by DCV$_2$ when the verifier knows that there is a single bad signature in the batch.

Consider the case when HV fails – this obviously indicates that the number of bad signatures is greater than 1. There are at least two possible courses of action:

1. Filter out all clean signatures identified by HV. Consider the syndrome string $(\sigma_1, \ldots, \sigma_k)$ generated by HV. Clearly, we can remove all clean sub-instances x_i for which $\sigma_i = 0$ and identify the bad signatures using DCV for the remainder of the batch.
2. Use the BCH code which corrects two errors to identify two bad signatures. This is an attractive option as we can reuse all results of GT tests obtained by HV. This gives rise to two level Hamming verifier defined below.

Unfortunately, BCH codes correcting two errors are not directly applicable. The main reason is different interactions of bad signatures compared to transmission errors in codes. Note that if two errors occur in a communication channel then they cancel each other in a parity check equation or more precisely, they obey the XOR addition. On the other hand, the behaviour of bad signatures is governed (with overwhelming probability) by logical addition. A parity checking equation failure does not depend on how many bad signatures it contains. This fact make the problem more difficult but also more interesting.

Assume that we have a batch of $n = 2^k - 2$ signatures which includes two bad ones ($t = 2$). As previously, we start from a Hamming code correcting a single error with the corresponding parity check matrix

$$H_1 = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \end{bmatrix} = \begin{bmatrix} 1 \; 2 \ldots 2^k - 2 \end{bmatrix} = \begin{bmatrix} 1\,0 \ldots 0 \\ 0\,1 \ldots 1 \\ \vdots \; \ldots \\ 0\,0 \ldots 1 \end{bmatrix}$$

Note that this matrix does not contain any column with all ones. We define another matrix

$$H = \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} \tag{12}$$

where H_1 is as defined above and H_2 is the negation of H_1, i.e. $H_2(i,j) = \overline{H_1(i,j)}$ for $i, j = 1, \ldots, k$.

Definition 8. Two-Layer Hamming Verifier. *Given a batch instance* $x = ((m_1, s_1), \ldots, (m_n, s_n))$ *of length* $n = 2^k - 2$ *for some positive* k *and a linear code represented by its parity check matrix* H *with* $2k$ *rows and* n *columns of the form given by Equation 12. Assume that the batch is contaminated by two bad signatures with their indices*

$$I_1 = (i_{1,1}, \ldots, i_{1,k}) \text{ and } I_2 = (i_{2,1}, \ldots, i_{2,k})$$

1. *Apply the generic test on the input instance. If* $GT(x, n)$*=0, exit. Otherwise, go to the next step.*
2. *Create* $2k$ *sub-instances (or control groups) corresponding to rows of the matrix* H *or*

$$x_{1,i} = \{(m_j, s_j) | H_1(i, j) = 1 \text{ and } j = 1, \ldots, n\}$$
$$x_{2,i} = \{(m_j, s_j) | H_2(i, j) = 1 \text{ and } j = 1, \ldots, n\}$$

   for $i = 1, \ldots, k$.
3. *Run* $GT(x_{1,i}, 2^{k-1} - 1) = \sigma_i$ *and* $GT(x_{2,i}, 2^{k-1} - 1) = \sigma'_i$ *for* $i = 1, \ldots, k$. *Create two syndromes* $\sigma = (\sigma_1, \ldots, \sigma_k)$ *and* $\sigma' = (\sigma'_1, \ldots, \sigma'_k)$.
4. *Identify an index* ℓ *such that both* $\sigma_\ell = 1$ *and* $\sigma'_\ell = 1$. *As the two corresponding control groups complement each other and both are contaminated, this implies that each group contains a single bad signature.*
5. *Run the HV verifier for* $x_{1,\ell}$ *and identify the bad signature. In result, the index* I_1 *is known.*
6. *Calculate the second index* $I_2 = I_1 \oplus \sigma \oplus \overline{\sigma'}$.
7. *Run the GT test for the input batch without the two bad signatures identified by indices* (I_1, I_2). *If the test accepts the batch, return the two indices, otherwise, the verifier fails and exits 1.*

Take a closer look at the two-Layer Hamming Verifier (2HV). All steps are straightforward except the part when the second index is computed. Note that the indices and syndromes satisfy the following equations:

$$I_1 + I_2 = \sigma$$
$$\overline{I_1} + \overline{I_2} = \sigma'$$

where + is a bit-by-bit logical OR. Note that + operation can be replaced by bit-by-bit XOR. Also the second equation can be converted using DeMorgan's Law. Thus

$$I_1 \oplus I_2 \oplus I_1 I_2 = \sigma$$
$$I_1 I_2 = \overline{\sigma'}$$

This allows us to determine the other index knowing the first as

$$I_2 = I_1 \oplus \sigma \oplus \overline{\sigma'} \tag{13}$$

Let us analyse the complexity of the 2HV verifier. Step (1) takes one GT test. Step (3) consumes $2k$ GT tests. The HV verifier employed in Step (5) requires $(k-1)+2$ GT tests. Step (7) makes the final GT test. Overall, the 2HV verifier runs in expense of $3k+3$ GT tests. So we can formulate the following conclusion.

Proposition 1. *Given a batch instance contaminated by two bad signatures. Then the 2HV verifier always correctly identifies them and consumes* $3k+3$ *GT tests.*

Consider the case when instead of two bad signatures, a batch instance is contaminated by a single bad signature. The 2HV verifier will still work correctly returning two $I_1 = I_2$ indices. This case can be easily identified as syndromes $\sigma = \overline{\sigma'}$. This will allow to skip Step (5) and save on GT tests. If there is a high probability of a single bad signature occurring, then it would be better to run the HV verifier first (perhaps with $n = 2^k - 2$) and if it fails re-use the results in the 2HV verifier.

Now consider a simple example. Let a batch instance contain $n = 2^4 - 2 = 14$ signatures ($k = 4$). Assume that bad signature occur on 6th (0110) and 10th (1010) positions. The linear code used is defined by its matrix H of the form

$$H = \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} = \begin{bmatrix}
1&0&1&0&1&\mathbf{0}&1&0&1&\mathbf{0}&1&0&1&0 \\
0&1&1&0&0&\mathbf{1}&1&0&0&\mathbf{1}&1&0&0&1 \\
0&0&0&1&1&\mathbf{1}&1&0&0&\mathbf{0}&0&1&1&1 \\
0&0&0&0&0&\mathbf{0}&0&1&1&\mathbf{1}&1&1&1&1 \\
&&&&&&&&&&&&& \\
0&1&0&1&0&\mathbf{1}&0&1&0&\mathbf{1}&0&1&0&1 \\
1&0&0&1&1&\mathbf{0}&0&1&1&\mathbf{0}&0&1&1&0 \\
1&1&1&0&0&\mathbf{0}&0&1&1&\mathbf{1}&1&0&0&0 \\
1&1&1&1&1&\mathbf{1}&1&0&0&\mathbf{0}&0&0&0&0
\end{bmatrix}$$

We create sub-instances according to Step (2) and compute syndromes $\sigma = (0111)$ and $\sigma' = (1011)$. Note that $\sigma_3 = \sigma'_3$ so the third control groups in H_1 and H_2 complement each other and contain single bad signatures. Now we apply the HV verifier for the third control group in H_1 and identify $I_1 = (0110)$. The second index is $I_2 = I_1 \oplus \sigma \oplus \overline{\sigma'} = (0110) \oplus (0111) \oplus (0100) = (0101)$.

6 General Model for Verification Codes

Consider the 2HV verifier. One would hope that the indices I_1 and I_2 could be identified using $2k$ tests which correspond to the control groups defined by the matrix H. Ideally, one would expect that from the two equations

$$\begin{aligned} I_1 \oplus I_2 \oplus I_1 I_2 &= \sigma \\ f(I_1) \oplus f(I_2) \oplus f(I_1)f(I_2) &= \sigma', \end{aligned} \tag{14}$$

it is possible to determine both I_1 and I_2. The function $f : \Sigma^k \rightarrow \Sigma^k$ is a Boolean function which for a given k-bit input, generates k-bit output. Now we prove that the following result is true.

Theorem 1. *Given four binary strings* $I_1, I_2, \sigma, \sigma' \in \Sigma^k$ *used in the 2HV verifier and satisfying Equation (14), then there is no function* $f : \Sigma^k \rightarrow \Sigma^k$ *for which the equations have unique solutions for* I_1 *and* I_2*.*

Proof. First observe that Equation (14) is satisfied if and only if it is true for each bit. The proof reduces to the binary case – instead of Equation (14) we consider its binary version

$$\begin{gathered} i \oplus j \oplus ij = u \\ f(i) \oplus f(j) \oplus f(i)f(j) = v, \end{gathered} \tag{15}$$

where $i, j, u, v \in \Sigma$. For the function $f : \Sigma \to \Sigma$, there are four possibilities only: $f(x) \in \{0, 1, x, \overline{x}\}$. The constant functions $f(x) = 0$ and $f(x) = 1$ are not an option. The only candidates are $f(x) = x$ and $f(x) = \overline{x}$ The results are given in Table 3. Consider the value u (3rd column) and the value v for $f(x) = x$ (6th column). If $u = v = 0$, there is the unique solution for $i = j = 0$. If $u = 0; v = 1$, there is no solution. For $u = v = 1$, there are three indistinguishable solutions. Consider the other function $f(x) = \overline{x}$ and the values u and v (9th column). If $u = 0$ and $v = 1$, there is unique solution $i = j = 0$. If $u = 1$ and $v = 1$, there is unique solution $i = j = 1$. For $u = v = 1$, there are two solutions $(i = 0, j = 1)$ and $(i = 1, j = 0)$. The combination $u = v = 0$ cannot occur.

Table 3. The truth table for two candidates of $f(x)$

i	j	u	$f(i) = i$	$f(j) = j$	v	$f(i) = \overline{i}$	$f(j) = \overline{j}$	v
0	0	0	0	0	0	1	1	1
0	1	1	0	1	1	1	0	1
1	0	1	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0

Although the above theorem gives us a "cold" comfort, it also points towards a different approach. Given a batch instance of $n = 2^k$ signatures with $t = 2$ bad ones. We are looking for a matrix H with n columns and ℓ rows $(l > 2k)$ such that any two indices I_1, I_2 (this time treated as the column vectors with ℓ bits) generate the unique result $I_1 + I_2$ (+ is bit-by-bit OR). In other words, we search for such an arrangement of rows of H that no two pairs of indices collide. The first question to be answered is the size of parameter ℓ for which a such arrangement may exist. If we assume that each column of the matrix H contains half of "1" then the parameter ℓ must satisfy the following inequality

$$\binom{\ell}{\frac{\ell}{2}} > \binom{n}{2}$$

It is easy to verify that for $k = 3$, $\ell \geq 2k + 1$. If k grows, then for $k = 40$, $\ell \geq 2k + 3$.

Definition 9. Generic Verifier (GV) *Given a batch instance x of length $n = 2^k$ for some positive k and a linear code represented by its parity check matrix*

H with ℓ rows ($\ell > 2k$) and n columns. Assume that the batch is contaminated by t bad signatures with their indices $I_1, \dots, I_t$ which are column vectors of H. The syndrome $\sigma = I_1 + \dots + I_t$ which uniquely identifies the indices $I_1, \dots, I_t$.

1. *Apply the generic test on the input instance. If $GT(x,n)$=0, exit. Otherwise, go to the next step.*
2. *Create ℓ sub-instances (or control groups) corresponding to rows of the matrix H.*
3. *Run ℓ times the GT test and form the syndrome σ.*
4. *Identify indices $I_1, \dots, I_t$ from the syndrome σ.*
5. *Run the GT test for the input batch without t bad signatures identified. If the test accepts the batch, return the t indices, otherwise, the verifier fails and exits 1.*

6.1 Verification Codes from Combinatorial Designs

Combinatorial designs provide an inexhaustible source of structures with unlimited potential for new designs of verification codes. We start from a simple and not efficient structure to show at least, in principle, that verification codes may be constructed from well known combinatorial designs [7].

Theorem 2. *Let D be the incidence matrix of a SBIBD(v,k,λ) where $v > 2k$, $k > 2\lambda$. Then D is a verification code allowing identification of any two bad signatures.*

Proof. Note that columns in the D matrix represent the control groups or sub-collection of signatures which are to be tested. By contradiction. Assume that a 2-SBIBD has two pairs of rows (B_1, B_2) and (B_3, B_4) such that

$$B_1 \cup B_2 = B_3 \cup B_4.$$

Without loss of generality, we can write the incidence matrix of D where the first k elements of the first row B_1 are ones and the remainder are zeros. We can also write D with the first λ elements of the second row B_2 to be "1", the next $k-\lambda$ elements to be "0", the next $k-\lambda$ elements – "1" and the remaining k ones in the first $2k-\lambda$ – "0", The next two rows (B_3 and B_4) have k ones in the first $2k-\lambda$ columns and the last $v-2k+\lambda$ elements zero since $B_1 \cup B_2 = B_3 \cup B_4$. Hence

$$D = \begin{bmatrix} \underbrace{1\cdots1\cdots1}_{k}\ \underbrace{0\cdots0}_{k-\lambda}\ \underbrace{0\cdots0}_{v-2k+\lambda} \\ \underbrace{1\cdots1}_{\lambda}\underbrace{0\cdots0}_{k-\lambda}\underbrace{1\cdots1}_{k-\lambda}\ 0\cdots0 \\ \underbrace{k \text{ ones}}_{2k-\lambda} \qquad 0\cdots0 \\ \underbrace{k \text{ ones}}_{2k-\lambda} \qquad 0\cdots0 \end{bmatrix}$$

Consider the ones in the first three rows, ensuring the inner product is λ. Suppose that t ones overlap with both the first and second row ($0 \leq t \leq \lambda$). Since the last $(v-2k+\lambda)$ columns contain only zeros, the number of ones in the third row is t in the first λ columns, $\lambda - t$ in the next $k - \lambda$ columns, and further $\lambda - t$ in the next $k - \lambda$ columns. Thus $t + 2\lambda - 2t = k$ and $t = 2\lambda - k \geq 0$. Hence $2\lambda \geq k$ – this is requested contradiction which proves the theorem.

7 Conclusions

Clearly, the above defined generic verifier sets the environment for the future research. In particular, the following list points out some open problems:

– lower bounds for the parameter ℓ or even better a function which determines the required parameter ℓ for a given t,
– how to design the matrix H so the syndrome uniquely identifies the indices (bad signature positions),
– how to design a verification code so identification of bad signatures is efficient,
– determine t for which GV becomes no better than NV,
– constructions of verification codes from combinatorial designs.

Acknowledgement

The authors wish to thank anonymous referees for their critical comments.

References

1. M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *Advances in Cryptology - EUROCRYPT'98*, pages 236–250. Springer, 1998. Lecture Notes in Computer Science No. 1403. 29, 30
2. M. Beller and Y. Yacobi. Batch Diffie-Hellman key agreement systems and their application to portable communications. In R. Rueppel, editor, *Advances in Cryptology - EUROCRYPT'92*, pages 208–220. Springer, 1993. Lecture Notes in Computer Science No. 658. 29
3. Elwyn Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, New York, 1968. 39
4. J-S. Coron and D. Naccache. On the security of RSA screening. In H. Imai and Y. Zheng, editors, *Public Key Cryptography – Second International Workshop on Practice and Theory in Public Key Cryptography, PKC'99*, pages 197–203. Springer, 1999. Lecture Notes in Computer Science No. 1560. 29, 32
5. L. Harn. Batch verifying multiple DSA-type digital signatures. *Electronics Letters*, 34(9):870–871, 1998. 30
6. D. Naccache, D. M'Raihi, S. Vaudenay, and D. Raphaeli. Can DSA be improved ? complexity trade-offs with the digital signature standard. In A. De Santis, editor, *Advances in Cryptology - EUROCRYPT'94*, pages 77–85. Springer, 1995. Lecture Notes in Computer Science No. 950. 29
7. A.P. Street and W.D. Wallis. *Combinatorics: A First Course*. CBRC, Winnipeg, 1982. 44
8. S. Yen and C. Laih. Improved digital signature suitable for batch certification. *IEEE Transactions on Computers*, 44(7):957–959, 1995. 29, 31