Attacking and Repairing Batch Verification Schemes

Colin Boyd and Chris Pavlovski

Information Security Research Centre
School of Data Communications
Queensland University of Technology
Brisbane, Australia
boyd@fit.qut.edu.au, chripavl@au1.ibm.com

Abstract. Batch verification can provide large computational savings when several signatures, or other constructs, are verified together. Several batch verification algorithms have been published in recent years, in particular for both DSA-type and RSA signatures. We describe new attacks on several of these published schemes. A general weakness is explained which applies to almost all known batch verifiers for discrete logarithm based signature schemes. It is shown how this weakness can be eliminated given extra properties about the underlying group structure. A new general batch verifier for exponentiation in any cyclic group is also described as well as a batch verifier for modified RSA signatures.

1 Introduction

Modular exponentiation is a fundamental operation for most practical digital signature schemes. The computational expense of both signing and verifying signatures is mainly due to the modular exponentiation required. Several techniques have been proposed in the literature to reduce this expense, including use of small exponents, and multi-exponentiation techniques [21]. An alternative way to realize a computational reduction is through use of batch cryptography.

Batch cryptography is relevant in settings where many signatures (or other primitives) need to be generated and/or verified together. Electronic commerce applications are prime examples, as typically many customers interact with the same merchant or banking server. Although techniques have been developed to improve signature generation [6,16], the majority of the recent work in the area has focused on the batch *verification* of signatures. These techniques all exploit the homomorphic properties of exponentiation in various groups to combine a set of exponentiations into one equation whose computational effort is effectively divided amongst all the individual exponentiations required.

The purpose of this paper is to illustrate flaws in a number of published batch verifiers; in some cases they are broken whilst in others we show that they do not provide the strength of verification claimed. We show that an observation of Bellare *et al.* [1], regarding the restrictions on use of certain batch verifiers, has much more serious consequences than they imply; in most applications this makes the tests ineffective. Through stronger assumptions on the group structure we show how these tests may be repaired.

1.1 Background

The idea of batch cryptography was introduced by Fiat [6,7]; his scheme amortized the private key operations for RSA and so was designed to assist in the signing and decryption operations. His idea was to batch a number of messages together, perform one full-scale modular exponentiation to sign the messages simultaneously, and then split apart the batch into individually signed messages. This is achievable due to the homomorphic property of RSA and the use of multiple, relatively prime, public exponents, an idea introduced by Chaum [4].

Batch verification for DSA signatures was introduced by Naccache, M'Raïhi, Raphaeli and Vaudenay [15]. Their scheme is designed to verify several DSA signatures at once by checking that a batch criterion holds and is much more efficient than sequential verification of individual DSA signatures[1]. Harn subsequently proposed a new method for DSA signatures requiring interaction between signer and verifier [10] and later devised a non-interactive version [11].

Early work concerning (non-interactive) batch verification was also published by Yen and Laih [22]. Their verification techniques are proposed for batch verification of a modification of the Schnorr or Brickell-McCurley signature schemes as well as for RSA. The principle, once again, is based upon the homomorphic properties of the respective scheme. Yen and Laih also note that to remain secure from attack, the verifier must choose random exponent values and apply these during batch verification. These values prevent the signer from attempting to introduce false signatures that would otherwise satisfy the batch verification criterion (the properties of this test are discussed in more detail in section 1.2).

Recently, Bellare, Garay, and Rabin [1,2] described several techniques for conducting batch verification of exponentiation with high confidence that false values have not been mixed into the batch. The technique which they refer to as the *small exponents test*, is very similar to the algorithms of Naccache *et al.* [15] and Yen and Laih [22], while their more sophisticated *bucket test* turns out to be more efficient for larger batch instances.

1.2 Batch Verification of Exponentiation

First we give a general idea of how batch verification of exponentiation works in a group. Consider the situation where we are given n elements $y_1, y_2, \ldots, y_n$, all in a multiplicative group G, and n exponents $x_1, x_2, \ldots, x_n$, all integers up to some size (we will become more specific shortly). A fixed element $g \in G$ is known. The idea of batch verification is to check that $y_i = g^{x_i}$ for each i without having to make this explicit calculation n times. In the case that the x_i values are indeed the discrete logarithms of the respective y_i values we will say that

[1] An earlier version of the paper of Naccache *et al.* included an additional interactive batch verifier. Lim and Lee [12] showed that this version is not secure.

the batch is *correct.* A good batch verification algorithm should identify, at least with high probability, whenever one or more of the x_i values is not the discrete logarithm of the respective y_i.

All the known batch verification techniques are based on the multiplicative property of the group. Specifically, if the batch is correct then the following equation holds.

$$\prod_{i=1}^{n} y_i = g^{\sum_{i=1}^{n} x_i} \tag{1}$$

It is easily checked that the converse is false: if equation 1 holds then it need not be the case that the batch is correct. For example, adding a constant to one x_i value and subtracting the same constant from a different x_i value does not change equation 1 but invalidates the batch. Another example is where the correct x_i values are randomly permuted.

Various authors [15,22] have noticed this and suggested that, to turn equation 1 into a useful batch verifier, randomisation should be introduced. This is done by multiplying the x_i values by small random values which must also be introduced as small exponents for the y_i values. An attacker who wishes to have an incorrect batch accepted has to anticipate which random values will be used. We follow Bellare *et al.* [1] and call this idea the *small exponents test.* The algorithm is shown in table 1. Bellare *et al.* prove that the small exponents test is a good batch verifier with error bounded by 2^{-l} as long as q, the order of the group G, is prime. It can be seen that the algorithm uses one full exponentiation in G plus n multiplications to obtain x and finally the cost of the n small exponentiations to find y. Bellare *et al.* use a multi-exponentiation algorithm to show that the total average cost is $l+n(1+l/2)$ multiplications in addition to the full exponentiation.

GIVEN: g a generator of the group G of prime order q, and $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_i \in \mathbb{Z}_q$ and $y_i \in G$. Also a security parameter l.
CHECK: That $\forall i \in \{1, \ldots, n\} : y_i = g^{x_i}$.

1. Pick $s_1, \ldots, s_n \in \{0,1\}^l$ at random.
2. Compute $x = \sum_{i=1}^{n} x_i s_i \bmod q$ and $y = \prod_{i=1}^{n} y_i^{s_i}$.
3. If $g^x = y$ then accept, else reject.

Table 1. Small exponents test for batch verification of exponentiation [1]

We will concentrate on the small exponents test in this paper. Bellare *et al.* also propose a variation which they call the *bucket test* which can be more efficient for large batches. Our general results apply also to the bucket test and we discuss the difference further in section 3.2.

A critical assumption in the small exponents test is that the y_i values lie in the group of prime order, G. This rules out the case where G is the multiplicative group $\mathbb{Z}_n^*$ for n composite as used in RSA and related algorithms. Nevertheless, Bellare *et al.* have shown that there is a simpler form of verification, which they called *screening*, that applies to RSA signatures[2]. Screening shows that the signatures must have, at some time, been formed by the true owner of the private key even though none of the individual claimed signatures might actually be correct. Screening is sufficient in applications where it is not necessary to possess the signatures, but only to know that the messages were signed; an example might be bulk verification of certificates.

1.3 Central Observation and Contribution

As mentioned above, it is a requirement in the proof of correctness of the small exponents test that all operations are performed within a group G of prime order. Bellare *et al.* suggest that in practice this is not really a restriction as this setting is commonplace in many modern cryptographic schemes.

They observe that when the order of G is not prime the small exponents test will not work. For an example they use $G = \mathbb{Z}_p^*$, which has non-prime order $p - 1$. Let g be a generator of $\mathbb{Z}_p^*$, and suppose $y = g^x \bmod p$. Under these assumptions the small exponents test will not detect the invalid batch with two pairs $(x, -y \bmod p), (x, y)$ when the small exponent for the first pair is even, which occurs with probability $1/2$. Notice that if y lies in some prime order subgroup G then $-y$ cannot lie in G.

The theme of this paper revolves around the requirement of working in a prime order group, and can be summarised in two significant observations.

1. Several authors have ignored this requirement. We give explicit attacks to show that their proposed batch verifiers do not work as advertised.
2. Even when this requirement is stated, it is not usually possible to check efficiently that it actually holds in a batch presented for verification. This makes most applications, including batch verification of DSA signatures [2,15], inappropriate unless additional properties hold.

The remainder of this paper is structured as follows. In the next section we show that the claimed strong RSA batch verifiers proposed by Yen and Laih [22] actually provide only the weaker screening property. We also present an explicit attack on the batch DSA verifiers of Harn [11], showing that an outsider can forge a batch signature for messages of his choosing. In the following section we outline a general attack that is applicable to verifiers of signatures in batches, illustrating how this may be applied to the small exponents test for batch verification of DSA signatures [2,15]. The attack allows the true signer to have false signatures accepted by the verifier. We then demonstrate how this general attack may be avoided by careful choice of the prime modulus used and give a generalised small exponents test for any cyclic group. We finally present a batch verifier for modified RSA signatures.

[2] Coron and Naccache [5] pointed out that screening can fail if duplicate messages are present. A modified version of screening was later proven correct by Bellare *et al.* [2].

2 Specific Attacks on Batch Verification Schemes

In this section we look at two schemes for batch verification which do not operate in prime order groups. The first works with a composite modulus, while the second performs a modular reduction before verification which destroys the group structure. We show that in both cases the verification does not provide the assurances claimed.

2.1 Yen and Laih's RSA Batch Verification

Yen and Laih [22] proposed a variation of ElGamal signatures suitable for batch signature verification. Here we consider the RSA batch verification technique that they devised as a performance comparison with their proposed scheme. They have essentially proposed to use the small exponents test in the RSA multiplicative group. Specifically, suppose that $S_1, \ldots, S_n$ are claimed RSA signatures [18] on messages $m_1, \ldots, m_n$ (where these messages have been pre-processed by any chosen hashing and redundancy functions). If the signatures are correct then $S_i = m_i^d \bmod N$ where d is the RSA private exponent and N the modulus. Small exponents $s_1, \ldots, s_n$ are chosen randomly of length l. The batch verification is then to test if the following equation holds, where e is the RSA public exponent.

$$\left(\prod_{i=1}^{n} S_i^{s_i} \right)^e \equiv \prod_{i=1}^{n} m_i^{s_i} \bmod N \tag{2}$$

Notice that this test is not as efficient as the small exponents test described in table 1 because it is not possible for the verifier to add the exponents on the left hand side modulo the group order. Furthermore, in practice a small value of e is often used which severely limits the benefit of batch verification. For example, if $e = 3$ then the batch verification can never be as efficient as individual verification of the signatures with any reasonable failure probability. But regardless of the test's efficiency it is wrong to assume that is provides more than screening; this means that use of the small exponents is redundant since Bellare *et al.* showed that equation 2 provides screening with all $s_i = 1$, at least in the case of full domain hashing.

The simplest attack is to replace some S_i values by $-S_i$ and some m_i values by $-m_i$ (all modulo N). Then the test will still succeed with probability 1/2 depending on the parity of the s_i values chosen. This attack can be launched by any party. It can be compounded by the signer who can choose an element α of small order t in the multiplicative group (t should be smaller than 2^l). Any S_i value can then be replaced by $\alpha S_i \bmod N$ and the test will succeed with probability $1/t$. Note that it is easy to find such an α if the factorisation of N is known.

2.2 Harn's DSA Batch Verification

Harn [11] proposed an algorithm which is essentially a direct application of equation 1 to variants of DSA signatures. Specifically he considers the following signature algorithm. Primes p and q are chosen with $q|p-1$ and a generator g of the group G of order q is published. A user's private key is a number x in $\mathbb{Z}_q$ and the corresponding public key is $y = g^x \bmod p$. A signature of a message m (again pre-processed by hashing) is a pair (r, s) where both r and s lie in $\mathbb{Z}_q$. A claimed signature pair is correct if the following verification equation holds.

$$r = (g^{sr^{-1} \bmod q} y^{mr^{-1} \bmod q} \bmod p) \bmod q$$

Now suppose that $m_1, \ldots, m_n$ is a batch of messages with corresponding set of claimed signatures $(r_1, s_1), \ldots, (r_n, s_n)$. Applying the multiplicative property, the following equation holds, which is also the proposed batch verification test.

$$\prod_{i=1}^{n} r_i \bmod q = (g^{\sum_{i=1}^{n} s_i r_i^{-1} \bmod q} y^{\sum_{i=1}^{n} m_i r_i^{-1} \bmod q} \bmod p) \bmod q \tag{3}$$

Our first observation is that this test can provide no more than screening. For suppose that a batch of correct signatures is known. Keep the r_i values the same and then choose the $n-1$ values $s'_1, \ldots, s'_{n-1}$ randomly and finally solve the equation

$$\sum_{i=1}^{n} s'_i r_i^{-1} \bmod q = \sum_{i=1}^{n} s_i r_i^{-1} \bmod q$$

to obtain the value s'_n. Then the batch $(r_1, s'_1), \ldots, (r_n, s'_n)$ satisfies the test but almost certainly none of the signatures is correct.

Now we show that the situation is compromised even further by an explicit attack. With high probability it is possible for an attacker who is not the signer to find signatures for any chosen message set. We only need to assume that the attacker has any known signature for this scheme: this gives values A, B and C with $A = (g^B y^C \bmod p) \bmod q$. We suppose that the attacker has chosen two messages for signing, say m_1 and m_2 (the attack is easily generalised to any number of messages). The attack works by making verification equation 3 the same as for the known signature. This is done in two steps.

1. Solve for r_1 and r_2 to ensure that
$$r_1 r_2 \equiv A \bmod q$$
$$m_1 r_1^{-1} + m_2 r_2^{-1} \equiv C \bmod q.$$
2. Solve for s_1 and s_2 to ensure that
$$s_1 r_1^{-1} + s_2 r_2^{-1} \equiv B \bmod q.$$

The simultaneous equations in step 1 can be reduced to the quadratic equation $(m_2/A) r_1^2 - C r_1 + m_1 \bmod q$ which can be solved by completion of the square as long as the discriminant $C^2 - 4m_1 m_2/A$ is a quadratic residue modulo q. On the assumption that m_1 and m_2 are random (they are the result of hashing) this will be the case with probability 1/2. Step 2 can then be completed by choosing s_1 randomly and solving for s_2.

The attack can be generalised for any number of messages to be forged. In step 1 all but two of the r_i values can be chosen randomly and then the remaining two found by solving a quadratic equation as described above. Step 2 proceeds as above with all but one of the s_i values chosen at random. It is interesting to note that this attack will not work if random small exponents are added to the verification equation. However, since there is no security proof it would be dangerous to rely on such a test.

3 General Attack on the Small Exponents Test

In this section we show that the small exponents test described in table 1 is much less useful that it at first appears. We will show that many of the proposed applications for the test are, in fact, not appropriate at all.

3.1 Attacking Batch Verification of DSA

In order to explain the weakness we first describe the batch DSA verification proposed by Bellare *et al.* [2]. (Note that this application was not included in the shortened version of the paper published at Eurocrypt'98 [1]). As previously suggested by Naccache *et al.* [15] the verification algorithm is applied not to the original DSA signature scheme but to a slightly altered version.

The setting is again in a subgroup G of $\mathbb{Z}_p^*$ of prime order q where a user's private key is $x \in \mathbb{Z}_q$ with public key $y = g^x \in G$. The signature of a (pre-processed) message m is a triple (λ, s, m) which satisfies the following verification equation, where $r = \lambda \bmod q$.

$$\lambda = g^{ms^{-1} \bmod q} y^{rs^{-1} \bmod q} \bmod p$$

The difference in original DSA is that λ is replaced by r, and the verification equation is reduced modulo q. This means that the original DSA signature is only twice the size of q instead of the size of q plus the size of p in the revised version. Since typical sizes of p and q would be 1024 and 160 bits respectively, this is a significant extra overhead which might be worthwhile for the computational gains of batch verification. Note that the modified version can easily be converted into an original DSA version at any time by replacing λ with r. Bellare *et al.* applied the small exponents test to a batch of modified DSA signatures as shown in table 2.

We now apply our main observation to the algorithm: at no time in the algorithm is it checked that the λ_i values are actually within the group G as they should be. Once this is observed it is straightforward to develop an attack. (In contrast to the attack in section 2.2 only the true signer can carry out this attack.) Similar to the attack on Yen-Laih's algorithm in section 2.1, the idea of the attack is to replace one or more λ_i values by $-\lambda_i$ and the signatures will be accepted with probability 1/2. Because the b_i values in the test depend on λ_i the attacking signer needs to choose λ_i first and then find s_i. Specifically the signer proceeds as follows to run the attack with one or more of the messages m_i.

GIVEN: Public parameters p, q, g a public key y and a batch of claimed signatures: $(\lambda_1, s_1, m_1), \ldots, (\lambda_n, s_n, m_n)$ with $s_i \in \mathbb{Z}_q$ and $\lambda_i \in G$. Also a security parameter l.
CHECK: That $\forall i \in \{1, \ldots, n\} : \lambda_i = g^{m_i s_i^{-1} \bmod q} y^{r_i s_i^{-1} \bmod q} \bmod p$.

1. For $i = 1, \ldots, n$ set $a_i = s_i^{-1} m_i \bmod q$ and $b_i = s_i^{-1} \lambda_i \bmod q$.
2. Pick $w_1, \ldots, w_n \in \{0,1\}^l$ at random.
3. Compute $A = \sum_{i=1}^n a_i w_i \bmod q$, $B = \sum_{i=1}^n b_i w_i \bmod q$, and $R = \prod_{i=1}^n \lambda_i^{w_i}$.
4. If $g^A y^B = R$ then accept, else reject.

Table 2. Small exponents test for batch verification of modified DSA [2]

1. Choose k_i randomly in $\mathbb{Z}_q$ and set $L_i = g^{k_i} \bmod p$.
2. Set $\lambda_i = -L_i \bmod p$, $r_i = \lambda_i \bmod q$ and $s_i = k_i^{-1}(m_i + x r_i) \bmod q$.
3. Present (λ_i, s_i, m_i) to the verifier as part of the batch.

It follows that

$$g^{m_i s_i^{-1} \bmod q} y^{r_i s_i^{-1} \bmod q} \bmod p = g^{k_i} \bmod p = L_i$$

and since $L_i^2 \equiv \lambda_i^2 \bmod p$ this will go undetected if the verifier chooses this w_i to be even which happens with probability 1/2.

As with the attack on Yen-Laih, it can be generalised by substituting $\lambda_i = \alpha L_i \bmod p$ for an element α with any order t where $t | p-1$ and $t \leq 2^l$. Usually there will be many such t values that can be chosen. Then the signature will be accepted with probability $1/t$.

We would like to emphasise that this does not invalidate the theorem proven by Bellare *et al.* regarding the security of their small exponents test since it is an assumption in table 1 that the y_i values are in the group G. Furthermore, strictly the application is correct as long as the λ_i values are in G, but this is not a reasonable assumption in practice.

3.2 Other Schemes Susceptible to the Attack

Several other published schemes make essentially the same unjustified assumption. An attack on the earlier DSA batch verification scheme of Naccache *et al.* [15] is identical to that proposed above. A similar attack on the batch verifier for a Schnorr signature variant proposed by Yen and Laih [22] is possible. Note that if all (or most) of the small exponents will be chosen to be odd, such as is suggested by Naccache *et al.*, the substitution should be made on an even number of λ_i values for the attack to succeed.

Another application that is vulnerable is a recent proposal for batch verification of coins in Brands' cash scheme [17]. In this proposal the merchant essentially uses the small exponents test during the payment protocol to verify a batch of coins together; a batch test is also used by the bank at deposit time. A possible consequence of the above attack is that a customer can frame a merchant since there is a high probability that the customer can have a bad coin accepted at payment time but that it will be rejected by the bank during deposit.

The alternative *bucket test* of Bellare *et al.* [1] is also vulnerable to the same attack, since it basically consists of a series of small exponent tests run on random partitions of the batch. However, in many instances it will detect the attack with much higher probability than the small exponents test. The bucket test uses an additional parameter m, repeats the partitioning $\lceil l/(m-1)\rceil$ times, and runs the small exponents test with parameter m in place of l. So a value λ_i replaced by $-\lambda_i$ will be detected with probability $1/2$ for every repetition, or $2^{-\lceil l/(m-1)\rceil}$ overall. This is still much worse than the claimed probability of failure of 2^{-l}.

4 Repairing the Small Exponents Test

An obvious way to prevent the attack is to check that the λ_i values in table 2 are indeed in G, as required by the small exponents test. However, there does not appear to be any way to do this that does not totally negate the computational savings of the test. For example, to test directly that $\lambda_i^q \bmod p = 1$ would require n extra exponentiations. Note that it is not sufficient to check, for example, that the product of the λ_i values are in G.

The main problem in ensuring that the proof still holds is to avoid elements of low order in the 'large group'. The element of order 2 is always present in $\mathbb{Z}_p^*$ so we have to accept that there may sign changes in a batch that passes the test. In this section we show that through judicious choice of p it is possible to avoid any other problems.

4.1 Dealing with Prime Order Subgroups

First of all we assume that p is chosen to be of the form $p-1 = 2rq$ where r and q are both primes. The modified form of the small exponents test is shown in table 3; the differences from that in table 1 are small but significant. In particular there is no assumption that the y_i values lie in G. A consequence of this difference is that exponentiations are only known to be correct up to a possible multiple of -1. This should be acceptable in most applications since it can always be corrected if a particular value is later found to be incorrect.

The computational cost of the modified test is identical to that of table 1. Using an improved algorithm for multiexponentiation, Bellare *et al.* [1] calculated the total cost of the test as $l + n(1 + l/2)$ multiplications plus the cost of the exponentiation. The exact cost will depend on the size of the values of p, q and l (as well as the algorithms used for exponentiation and multi-exponentiation). Reasonable values today might be $|p| = 1024$, $|q| = 160$ and $l = 60$.

Theorem 1. *Suppose p is a prime and G a subgroup of $\mathbb{Z}_p^*$ of prime order q. If $p-1 = 2qr$ where r is prime and $min(q,r) > 2^l$ then the algorithm in table 3 is a batch verifier which fails with probability at most 2^{-l}.*

GIVEN: g a generator of the subgroup G of $\mathbb{Z}_p^*$ and $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_i \in \mathbb{Z}_q$ and $y_i \in \mathbb{Z}_p^*$. Also a security parameter l.
CHECK: That $\forall i \in \{1, \ldots, n\} : \pm y_i = g^{x_i}$.

1. Pick $s_1, \ldots, s_n \in \{0, 1\}^l$ at random.
2. Compute $x = \sum_{i=1}^n x_i s_i \bmod q$ and $y = \prod_{i=1}^n y_i^{s_i}$.
3. If $g^x = y$ then accept, else reject.

Table 3. Modified small exponents test for batch verification of exponentiation in $\mathbb{Z}_p^*$

Proof The proof is basically similar to that of Bellare *et al.* for their small exponents test but there are a few extra problems to consider. Suppose that g_0 is a generator of $\mathbb{Z}_p^*$ and suppose, without loss of generality, that $g = g_0^{2r}$. We can then write $y_i = g_0^{x'_i}$ for some x'_i with $1 \leq x'_i \leq p - 1$. Suppose that the test passes; then the following equation holds.

$$g_0^{2rx \bmod p-1} = g_0^{\sum_{i=1}^n x'_i s_i \bmod p-1}$$

Because g_0 is a generator of $\mathbb{Z}_p^*$ we have

$$2r(s_1 x_1 + \ldots + s_n x_n) \equiv x'_1 s_1 + \ldots + x'_n s_n \bmod (p-1)$$

which we may re-write as the following.

$$s_1(x'_1 - 2rx_1) + \ldots + s_n(x'_n - 2rx_n) \bmod (p-1) = 0 \tag{4}$$

Suppose that for at least one value of i we have $\pm y_i \neq g^{x_i}$. Without loss of generality let us assume that $i = 1$. If we suppose that the values of $s_2, \ldots, s_n$ have been chosen, then equation 4 is a linear equation in s_1 and the number of solutions for s_1 is either 0 or $\nu = (p-1, 2rx_1 - x'_1)$. Because $p - 1 = 2qr$, ν can take any of the eight values $\{1, 2, q, r, 2r, 2q, qr, 2qr\}$. But the case $\nu = 2qr$ means that $2rx_1 \equiv x'_1 \bmod p-1$ so $y_i = g^{x_i}$ which we have assumed is not true.

The next largest case is $\nu = qr$, so that we have either $2rx_1 \equiv x'_1 \bmod p-1$ or $2rx_1 + qr \equiv x'_1 \bmod p-1$. The former possibility is ruled out and the latter possibility means that $y_1 = g_0^{x'_1} = g_0^{2rx_1+qr} = -g^{x_1}$ which is also assumed not to hold.

The remaining cases do not satisfy the check so we need to show that they occur with small probability. The next largest case is $\nu = 2r$. Although in this case there are many solutions to equation 4, these solutions are evenly distributed in the sense that if X is any solution for s_1 then $X + q$ is also a solution. This means that there is at most one solution for s_1 in the range $0 \leq s_1 \leq 2^l$ since $q > 2^l$. A similar argument holds for all other possible value of ν. Since s_1 is chosen randomly the probability that equation 4 holds when $\pm y_1 \neq g^{x_1}$ is thus at most 2^{-l}. The same is then true if all $s_1, \ldots, s_n$ are drawn independently and randomly. □

It can be seen from the proof that the requirement that $p-1 = 2rq$ is stronger than necessary. In fact it is necessary only that $p-1$ has no factors smaller than q apart from 2. Efficient methods to generate primes satisfying either of these conditions have been described by Lim and Lee [13]. They suggest that to satisfy $p-1 = 2rq$, the prime r should be chosen first and then random primes q of the desired size chosen until p is prime. For $|p| = 1024$ only around 710 trials for q will be required which is a very practical requirement.

4.2 Generalisation and Applications

There are a number of ways that the modified small exponents test can be extended. We give a generalised form in table 4 which applies to any cyclic group. This algorithm can only give assurance of the correctness of the batch up to multiplication by an element of order less than 2^l. Therefore, in applications it will be useful to ensure that the group order has as few small factors as possible. The following theorem shows that the algorithm is a correct batch verifier. The proof, which is a generalisation of the proof of Theorem 1, is omitted due to space restrictions.

Theorem 2. *The algorithm in table 4 is a batch verifier which fails with probability at most* 2^{-l}.

GIVEN: g a generator of a cyclic group H of order ω and $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_i \in \mathbb{Z}_\omega$ and $y_i \in H$. Also a security parameter l.
CHECK: That $\forall i \in \{1, \ldots, n\} : \alpha y_i = g^{x_i}$ for some element $\alpha \in H$ of order less than 2^l.

1. Pick $s_1, \ldots, s_n \in \{0,1\}^l$ at random.
2. Compute $x = \sum_{i=1}^{n} x_i s_i \bmod \omega$ and $y = \prod_{i=1}^{n} y_i^{s_i}$.
3. If $g^x = y$ then accept, else reject.

Table 4. Generalised small exponents test for batch verification of exponentiation in any cyclic group

There are a number of useful applications of our modified small exponents tests.

– DSA batch verification can be achieved by adapting the algorithm of table 2 to verify the signature up to multiplication of each λ_i by -1. The algorithm is identical except
  - we require that $p-1$ has no factors smaller than 2^l apart from 2.
  - we do not assume that λ_i is in a prime order subgroup.

  Of course the attack in section 3.1 still holds so if this verification is to be used it is necessary to adapt the DSS algorithm so that (r, s, m) will be a correct signature if either of the following checks passes.

$$r = (g^{ms^{-1} \bmod q} y^{rs^{-1} \bmod q} \bmod p) \bmod q \quad (5)$$

$$(p - r) \bmod q = (g^{ms^{-1} \bmod q} y^{rs^{-1} \bmod q} \bmod p) \bmod q \quad (6)$$

Although it seems intuitively reasonable that this extension to DSA signatures is as secure as original DSA we do not offer any proof.

- Bellare *et al.* [1] have asked whether a batch verifier for exponentiation can be found for $\mathbb{Z}_p^*$ rather than in a prime order subgroup. The algorithm in table 4 answers this question in the affirmative (up to multiplication by -1) with the condition that $p - 1$ has no small factors apart from 2.
- The *bucket test* of Bellare *et al.* [1] is an extension of the small exponents test and it is immediate to extend our test in the same way. The computational cost will be the same as that of the original bucket test.

5 Batch Verification of RSA Signatures

As mentioned previously, Bellare *et al.* introduced screening as a weaker form of verification for RSA signatures. In this section we will use the ideas from the previous section to derive batch verification of a slightly modified definition of RSA signatures. This variation was already used by Gennaro *et al.* [8] in a different context. Specifically, the set of signatures on a message m, randomised appropriately, is defined as

$$\mathrm{SIG}(m) = \{S_m : S_m = \alpha m^d, ord(\alpha) \leq 2\}.$$

For an RSA modulus, $N = pq$, there are four possible signatures of every message. In addition to 1 and -1 there are two 'non-trivial' square roots of unity and knowledge of either of these allows N to be factorised. Consequently an oracle to forge a signature in $\mathrm{SIG}(m)$ can be used to forge an ordinary RSA signature either directly or by allowing factorisation of N.

The next restriction we need is that N should be the product of two safe primes: $N = pq$ where $(p - 1)/2$ and $(q - 1)/2$ are also prime. Since there is an efficient method to prove that N is of this form [3,14] this property can be checked when the public key is certified or, if necessary, prior to the batch verification. Our batch verification algorithm for RSA is given in table 5. The proof of the following result is omitted due to space restrictions.

Theorem 3. *The algorithm in table 5 is a batch verifier which fails with probability at most* 2^{-l}. *Its cost is approximately* $l(n+2)+1.5|e|+n-1$ *multiplications modulo* N.

Batch verification of RSA is counter-productive when e is small, as the cost of conventional sequential verification of the n signatures will be $1.5n|e|$ multiplications. The algorithm is worthwhile when e satisfies

$$|e| \gg \frac{l(n+2)}{1.5(n-1)} + \frac{2}{3}.$$

GIVEN: A modulus N which is the product of two primes and $(S_1, m_1), (S_2, m_2), \ldots, (S_n, m_n)$ with $S_i, m_i \in \mathbb{Z}_N$. Also a security parameter l with $2^l < \min(p', q')$.
CHECK: That $\forall i \in \{1, \ldots, n\} : \alpha S_i^e = m_i$ for some element $\alpha \in \mathbb{Z}_n^*$ of order not more than 2.

1. Check that $(S_i, N) = 1$ for all i.
2. Pick $s_1, \ldots, s_n \in \{0,1\}^l$ at random.
3. Compute $x = \left(\prod_{i=1}^n S_i^{s_i}\right)^e \bmod N$ and $y = \prod_{i=1}^n m_i^{s_i} \bmod N$.
4. If $x = y$ then accept, else reject.

Table 5. Small exponents test for batch verification of RSA signatures

Thus for large n we require that $|e| \approx 2l/3$ before the test becomes useful. Small values of e such as 3 or $2^{16} + 1$ will never benefit from our batch verification. There are certain situations where a random, or large, e is desirable [19]. For a random e our test provides immediate gains for any reasonable size of N.

6 Conclusion

In this paper we have outlined several new attacks on batch verification techniques in the literature including a general attack on batch verification which affects most of the prominent schemes. We have shown how this attack may be avoided by careful choice of the modulus and weakening the acceptance condition. We have also provided a new batch verifier for exponentiation in any cyclic group and a batch verifier for modified RSA signatures. These results answer many of the open questions posed by Bellare *et al.* [1].

Acknowledgements

We are grateful to Wenbo Mao for a number of helpful comments, and to the anonymous referees for constructive advice on presentation of the results.

References

1. M. Bellare, J. A. Garay, T. Rabin, Fast Batch Verification for Modular Exponentiation and Digital Signatures, Proceedings of Eurocrypt'98, LNCS, Vol. 1403, pp.236-250, Springer-Verlag, 1998.
2. M. Bellare, J. A. Garay, T. Rabin, Fast Batch Verification for Modular Exponentiation and Digital Signatures, available online at `http://www-cse.ucsd.edu/users/mihir`.
3. J.Camenisch and M.Michels, Proving in Zero-Knowledge that a Number is the Product of Two Safe Primes, Advances in Cryptology – Eurocrypt'99, Springer-Verlag, pp.107-122, 1999.

4. D. Chaum, A. Fiat, M. Naor, Untraceable electronic cash, Proceedings of Crypto '88, pp.319-227, 1988.
5. J.-S. Coron, D. Naccache, On The Security Of RSA Screening, Proceedings of PKC'99, Springer-Verlag, 1999.
6. A. Fiat, Batch RSA, Advances in Cryptology – Crypto '89, LNCS, Vol. 435, Springer-Verlag, pp.175-185, 1990.
7. A. Fiat, Batch RSA, Journal of Cryptology, 10, 2, pp.75-88, 1997.
8. R. Gennaro, H. Krawczyk and T. Rabin, RSA-Based Undeniable Signatures, Advances in Cryptology – Crypto'97, Springer-Verlag, 1997, pp.132-149.
9. A.J.Menezes, P.C. van Oorschot and S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.
10. L. Harn, DSA Type Secure Interactive Batch Verification Protocols, Electronics Letters, 31, 4, pp.257-258, 16th February 1995.
11. L. Harn, Batch Verifying Multiple DSA-type Digital Signatures, Electronics Letters, 34, 9, pp.870-871, 30th April 1998.
12. C.H. Lim and P.J. Lee, Security of Interactive DSA Batch Verification, Electronics Letters, 30, 19, pp.1592-1593, 15th September, 1994.
13. C.H. Lim and P.J. Lee, A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroup, Advances in Cryptology – Crypto'97, Springer-Verlag, 1997, pp. 249-262.
14. T.V. Le, K.Q. Nguyen and V. Varadharajan, How to Prove That a Committed Number is Prime, Advances in Cryptology – Asiacrypt'99, Springer-Verlag, 1999, pp.208-218.
15. D. Naccache, D. M'Raïhi, D. Raphaeli, S. Vaudenay, Can DSA be improved: complexity trade-offs with the digital signature standard, Proceedings of Eurocrypt'94, pp.85-94, 1994.
16. D. M'Raïhi and D. Naccache, Batch Exponentiation – A Fast DLP-Based Signature Generation Strategy, 3rd ACM Conference on Computer and Communications Security, pp.58-61, 1996.
17. C. Pavlovski, C. Boyd, E. Foo, Detachable Electronic Coins, Proceedings of ICICS '99, LNCS, Vol. 1726, Springer-Verlag, pp.54-70, 1999.
18. R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public key cryptosystems, Comm. ACM, pp.120-126, Vol. 21, No. 2, 1978.
19. H.-M. Sun, W.-C. Yang and C.-S. Laih, On the Design of RSA with Short Secret Exponent, Advances in Cryptology - Asiacrypt'99, Springer-Verlag, 1999, pp.150-164.
20. Y. Yacobi, M. Beller, Batch Diffie-Hellman Key Agreement Systems and their Application to Portable Communications, Proceedings of Eurocrypt 92, Vol. 658, pp.208-217, 1992.
21. S.M.Yen, C.S.Laih and A.K.Lenstra, Multi-exponentiation, IEE Proceedings, Part E: Computers and Digital Techniques, 141, 6, pp.325-326, 1994.
22. S. Yen, C. Laih, Improved Digital Signature Suitable for Batch Verification, IEEE Transactions on Computers, Vol. 44, No. 7, pp.957-959, July 1995.