Batch Verification for Equality of Discrete Logarithms and Threshold Decryptions

Riza Aditya[1], Kun Peng[1], Colin Boyd[1], Ed Dawson[1], and Byoungcheon Lee[1,2]

[1] Information Security Research Centre,
Queensland University of Technology,
GPO BOX 2434, Brisbane, QLD, 4001, Australia
{r.aditya, k.peng, c.boyd, e.dawson, b6.lee}@qut.edu.au
[2] Joongbu University,
101 Daebak-Ro, Chuboo-Meon, Kumsan-Gun, Chungnam, 312-702, Korea
sultan@joongbu.ac.kr

Abstract. A general technique of batch verification for equality of discrete logarithms is proposed. Examples of batching threshold decryption schemes are presented based on threshold versions of ElGamal and RSA cryptosystems. Our technique offers large computational savings when employed in schemes with a large number of ciphertexts to be decrypted, such as in e-voting or e-auction schemes using threshold decryption. The resulting effect is beneficial for producing more efficient schemes.

Keywords: Batch verification, Proof of equality of discrete logarithms (PEQDL), Threshold decryption, Threshold ElGamal, Threshold RSA

1 Introduction

Threshold decryption [14,18,11] is essential in fault-tolerant schemes, whether it is e-commerce (e.g: e-auction) or e-government (e.g: e-voting). In a threshold decryption protocol, the public (encryption) key is published, while the corresponding private (decryption) key is shared among n participants. A threshold $t < n$ is set, such that more than t participants are required to cooperate to decrypt a ciphertext while the cooperation of no more than t participants will find no information about the decryption key. In e-auction, these participants are auctioneers sharing the power to open the bids. In e-voting, the participants are counting authorities sharing the power to tally the votes.

To ensure correctness, it is necessary to guarantee that the shared decryption is performed correctly through some public verification functions, without revealing the encrypted message, the private key, and its shares. In many popular cryptosystems, the verification process is implemented by using zero-knowledge proof of equality of discrete logarithms (**PEQDL**) [6].

M. Jakobsson, M. Yung, J. Zhou (Eds.): ACNS 2004, LNCS 3089, pp. 494–508, 2004.

1.1 Performance Issue

Consider a secure e-auction [1,16] or e-voting scheme [2,4,8,11], where the submitted bids or ballots are required to be anonymous. After the encrypted bids or ballots are made anonymous through the use of an anonymous channel (e.g: mix network), they are decrypted by the decryption authorities. Each decryption share requires a proof of correct decryption. The proof is required to verify and identify correct decryption shares to reconstruct the original bid or vote. Verification of many instances of such proofs leads to costly computation which can further develop into a bottleneck affecting the performance of the scheme.

Batching is a useful technique to decrease computational cost in processing the proofs of correct decryption together. Bellare *et al.* [3] proposed three batch verification techniques - RS (random subset) test, SE (small exponent) test, and Bucket test. However, their scheme is not applicable to our threshold decryptions verification problem for two reasons. Firstly, their techniques batch the verification of common base exponentiations, not the verification of PEQDLs (i.e: common exponent). Secondly, Boyd and Pavlovski [5] demonstrated that although the theorems in [3] are correct, their application in the paper is inappropriate since the assumptions on the group structure are not strong enough. Hoshino *et al.* [12] later fixed and extended Bellare's work to batch verify exponentiations in multiple bases. However, this is also irrelevant to our problem of batch verifying PEQDLs with a common exponent.

1.2 Main Contributions

The following summarises our main contribution presented in the paper:

1. We fix the problem presented by Boyd and Pavlovski in SE test, and also extend the test to batch verify PEQDLs.
2. We present and formally prove theorems on the extended test.
3. We present applications of the theorems to verify valid decryption shares in threshold versions of two popular cryptosystems - threshold ElGamal, and threshold RSA.

Our result improves computational efficiency of verifying valid decryption shares in threshold decryption. As threshold decryption is fundamental in various applications (e.g: e-auction, e-voting, e-cash) to provide robustness, our result offers improvement in efficiency, performance and practicality when integrated with many schemes.

The remainder of this paper is structured as follows. Section 2 offers an introduction to threshold decryption. Section 3 presents two theorems and their corresponding proofs essential to our result. In section 4, the theorems are applied to threshold versions of the two popular cryptosystems. Sections 5 and 6 analyse the security and efficiency of our applied batch verification. Section 7 is a conclusion.

2 Background

In this section we recall decryption of a single ciphertext in threshold decryption schemes for simplicity. Note that many schemes require decryption of many ciphertexts in threshold decryption.

2.1 Threshold Decryption

In a threshold decryption scheme, a secret s is encrypted using some public-key encryption algorithm as $c = E(s)$. The private decryption key d is shared by using Shamir's (t, n) secret sharing scheme [17] among n participants (decrypting authorities) P_i, for $i \in \{1, \ldots, n\}$. Each P_i holds a d_i, a share of d. The ciphertext c is partially decrypted by each P_i as $z_i = D_{d_i}(c)$, and later reconstructed using the decryption shares from the set S containing at least $t+1$ honest participants by Lagrange interpolation.

A verification function $V(c, z_i, v_i)$ is used to determine honest participants. Normally the verification key v_i of participant P_i contains a commitment to d_i.

Threshold decryption is often employed in many crypto-based applications. The two most commonly used are threshold versions of ElGamal and RSA algorithms. E-auction and e-voting schemes employing them include [11,2,4,8,1, 16].

2.2 Threshold ElGamal

Pedersen [14] presented a threshold ElGamal signature scheme. It is straightforward to adjust the scheme into a threshold decryption protocol. We recall the protocol as follows:

1. Key generation and sharing:
   Randomly select a large prime q, such that $p = 2q + 1$ is also a prime. G is a cyclic subgroup in $\mathbb{Z}_p^*$ of order q with a generator g. The private decryption key is $d \in \mathbb{Z}_q$, while g and $e = g^d$ is the public encryption key. Using Shamir's secret sharing scheme, let $f(x) = \sum_{r=0}^{t} a_r x^r$, where $a_0 = d$, and the rest of a_r are random values. For $i \in \{1, \ldots, n\}$, distribute the secret share $d_i = f(i)$ to n participants $\{P_i\}$, and each P_i computes the verification key $v_i = g^{d_i}$. The parameters p, q, g, e and v_i are made public, while d and d_i are kept secret for $i \in \{1, \ldots, n\}$.
2. Encryption:
   Select a random $r \in \mathbb{Z}_q$ and encrypt a secret message $s \in \mathbb{Z}_p^*$ as a pair (α, β), where $\alpha = g^r$ and $\beta = se^r$.
3. Shared decryption:
   Each participant P_i computes the decryption share $z_i = \alpha^{d_i}$ and proves the knowledge of the secret share d_i using non-interactive zero-knowledge that:

$$\log_g(v_i) = \log_\alpha(z_i) \tag{1}$$

   Since q is public, g and α can be publicly verified to be generators of G.

4. Shares combining:
   Correct decryption share z_i of P_i is verified as P_i proves the knowledge of d_i shown in the previous step. S is the set of more than $t+1$ participants providing correct shares. The original message is reconstructed by computing $s = \frac{\beta}{\prod_{i \in S} z_i^{\mu_i}}$, where $\mu_i = \prod_{i' \in S, i' \neq i} \frac{i'}{i'-i}$.

2.3 Threshold RSA

Shoup [18] presented a threshold version of RSA signature scheme, which can be adjusted to a threshold decryption scheme as shown by Fouque *et al.* [11]. We recall the scheme as follows:

1. Key generation and sharing:
   Randomly select primes p' and q', such that $p = 2p' + 1$ and $q = 2q' + 1$ are strong primes. Set $N = pq$ and $M = p'q'$. Select a prime $e > n$ and compute d, such that $ed = 1 \bmod N$. The public encryption key is $PK = (N, e)$, while d is the private decryption key. Using Shamir's secret sharing scheme, let $f(x) = \sum_{r=0}^{t} a_r x^r \bmod M$, where $a_0 = d$ and random values for rest of $a_r \in \{0, \ldots, N * M - 1\}$. For $i \in \{1, \ldots, n\}$, distribute the secret share $d_i = f(i)$ to n participants P_i. Randomly select a verification base v in the cyclic group of squares in $\mathbb{Z}_N^*$. Each participant P_i then computes the verification key $v_i = v^{d_i} \bmod N$. The parameters N, e, v and v_i are made public, while M, p, q, p', q', d and d_i are kept secret, for $i \in \{1, \ldots, n\}$.
2. Encryption:
   Encrypt a secret message s as $c = s^e \bmod N$.
3. Shared decryption:
   Each participant P_i computes the decryption share $z_i = c^{2\Delta d_i}$, where $\Delta = n!$ and proves the knowledge of the secret share d_i using non-interactive zero-knowledge that:

$$\log_v(v_i) = \log_{c^{4\Delta}}(z_i^2) \tag{2}$$

   Notice that as v and $c^{4\Delta}$ are squares, Shoup argues that they are of order M with a large probability (accurately: $1 - \frac{p'+q'-1}{pq}$). Thus, the proof is assumed to be PEQDL in a group with a known order.
4. Shares combining:
   Correct decryption share z_i of P_i is verified as P_i proves the knowledge of d_i shown in the previous step. S is the set of more than $t+1$ participants providing correct shares. The original message is obtained by first calculating $s^{4\Delta^2} = \prod_{i \in S} z_i^{2\Delta\mu_i} \bmod N$, where $\mu_i = \prod_{i' \in S, i' \neq i} \frac{i'}{i'-i}$. Since $e > n$ is relatively prime to $4\Delta^2$, extended Euclidean algorithm can be applied to obtain a and b, such that $a \times 4\Delta^2 + b \times e = 1$. Therefore, s is reconstructed as $s = s^{a4\Delta^2} s^{be} = (s^{4\Delta^2})^a c^b \bmod N$.

As in the original scheme [18,11], parameters in the key generation and sharing stage are generated by a trusted dealer. The random verification base v is trusted to be in the cyclic subgroup of squares in $\mathbb{Z}_N^*$. Therefore, v and v_i are

squares in the group of $\mathbb{Z}^*_{N^2}$. As a result, when verification of Equation 2 is performed to check the validity of the decryption share, it is guaranteed to be PEQDL in the same cyclic group with a large probability.

3 Batch Verification for Equality of Logarithms

In many cryptographic applications as mentioned in the previous sections, normally there are many ciphertexts (c_j) to be processed in threshold decryption. This is illustrated in Figure 1. For m encrypted messages to be decrypted by n authorities, one requires $m \times n$ instances of PEQDL verifications of decryption share $z_{i,j}$ (participant i's decryption share from ciphertext c_j). Verification of correct shared decryption for every share $z_{i,j}$ is the greatest factor contributing to computational cost in a threshold decryption scheme.

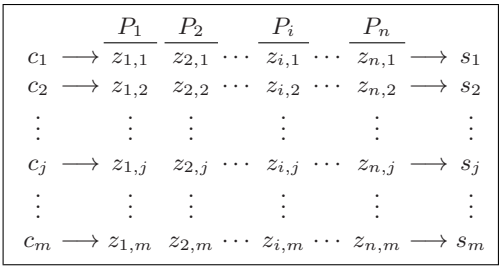

Fig. 1. Threshold decryption of n participants $\{P_i\}$, m ciphertexts $\{c_j\}$, mn decryption shares $\{z_{i,j}\}$, recovering m secret messages $\{s_j\}$

Techniques presented in [3], [5] and [12] only address batch verification for modular exponentiation. However, tests in [3] can be modified and extended to batch verify PEQDL. Hence, the efficiency of the threshold decryption scheme, as discussed in the previous paragraph, can be greatly improved.

This section presents two theorems on the modified SE test to batch verify PEQDL, i.e: verifying common exponent. Batching verification of common base is also briefly discussed. In Section 4, the theorems are used as a foundation to the applications proposed.

RS test randomly selects subsets of the instances to be verified in avoiding "bad pairs". This test is not sufficiently efficient, and thus is not discussed in this paper. SE test introduces random small exponents on the instances, such that an attacker needs to guess the random values to produce an accepted incorrect batch. This test is more suitable for our purpose and we modify this test on batch verification for PEQDL. Bucket test forms groups of the instances to be batched, and performs random SE tests on them. Our SE test can be extended naturally to Bucket test for batch verifying PEQDL. However, the extension of SE test to Bucket test for batch verifying PEQDL is omitted for simplicity. In the theorems below, we batch the verification of j instances of PEQDL on one participant ($i = 1$), and omit the subscript i.

3.1 Batching PEQDL within the Same Cyclic Group

Theorem 1 provides the foundation for batching PEQDL within the same cyclic group.

Theorem 1. *For $j \in \{1,\ldots,m\}$, G is a cyclic group with q as the smallest factor of $ord(G)$, generators g and c_j, and a security parameter l, where $2^l < q$. The small exponents t_j are random l-bit strings, and $y, z_j \in G$. If $\exists k \in \{1,\ldots,m\} \wedge \log_g y \neq \log_{c_k} z_k$, then $\log_g y \neq \log_{\prod_{j=1}^m c_j^{t_j}} \prod_{j=1}^m z_j^{t_j}$ with a probability (taken over choice of t_j) of no less than $1 - 2^{-l}$.*

To prove Theorem 1, we first prove the following lemma:

Lemma 1. *If $\exists k \in \{1,\ldots,m\} \wedge \log_g y \neq log_{c_k} z_k$, given a definite set $S = \{t_j | t_j < 2^l \wedge j \in \{1,\ldots,k-1,k+1,\ldots,m\}\}$, then there is only at most one t_k satisfying $\log_g y = \log_{\prod_{j=1}^m c_j^{t_j}} \prod_{j=1}^m z_j^{t_j}$, where $j \in \{1,\ldots,m\}$.*

Proof (Lemma 1). If the lemma is incorrect, the following two equations are satisfied simultaneously where $\log_g y \neq \log_{c_k} z_k$ and $t_k \neq t'_k$.

$$\log_g y = \log_{\prod_{j=1}^m c_j^{t_j}} \prod_{j=1}^m z_j^{t_j}$$

$$\log_g y = \log_{(\prod_{j=1}^{k-1} c_j^{t_j})(c_k^{t'_k})(\prod_{j=k+1}^m c_j^{t_j})} (\prod_{j=1}^{k-1} z_j^{t_j})(z_k^{t'_k})(\prod_{j=k+1}^m z_j^{t_j})$$

Suppose $y = g^x$, we re-write the two previous equations as:

$$(\prod_{j=1}^m c_j^{t_j})^x = \prod_{j=1}^m z_j^{t_j}$$

$$((\prod_{j=1}^{k-1} c_j^{t_j})(c_k^{t'_k})(\prod_{j=k+1}^m c_j^{t_j}))^x = (\prod_{j=1}^{k-1} z_j^{t_j})(z_k^{t'_k})(\prod_{j=k+1}^m z_j^{t_j})$$

Without losing generality, suppose $t'_k > t_k$, we can simplify the previous two equations to be $c_k^{x(t'_k - t_k)} = z_k^{t'_k - t_k}$, or $(\frac{c_k^x}{z_k})^{t'_k - t_k} = 1$. As $\frac{c_k^x}{z_k} \in G$, $t'_k - t_k$ is a factor of $ord(G)$ if $\frac{c_k^x}{z_k} \neq 1$. Since $0 < t'_k - t_k < q$, therefore, $\frac{c_k^x}{z_k} = 1$ or $c_k^x = z_k$. This is contradictory to the assumption of $\log_g y \neq \log_{c_k} z_k$. □

Proof (Theorem 1). Lemma 1 means that among the $(2^l)^m$ possible combinations of t_j for $j \in \{1,\ldots,m\}$, at most $(2^l)^{m-1}$ of them can satisfy $\log_g y = \log_{\prod_{j=1}^m c_j^{t_j}} \prod_{j=1}^m z_j^{t_j}$ when $\log_g y \neq \log_{c_j} z_j$. Therefore, given a random t_j for $j \in \{1,\ldots,m\}$, if $\log_g y \neq \log_{c_j} z_j$, then $\log_g y = \log_{\prod_{j=1}^m c_j^{t_j}} \prod_{j=1}^m z_j^{t_j}$ is accepted with a probability of no more than 2^{-l}. □

3.2 Batching PEQDL in Different Cyclic Subgroup of $\mathbb{Z}_p^*$

In Theorem 1, there is a condition that $g, y, c_j, z_j \in G$ for $j \in \{1, \ldots, m\}$. However, in some applications there is uncertainty of satisfaction on this condition, and additional computation is often required to verify the condition. This is a problem ignored by Bellare *et al.* [3]. In reality, this extra computation is too expensive so that in many cases it prevents the applicability of Theorem 1.

To overcome this problem, Theorem 2 is proposed. This theorem does not require the pre-condition that the LHS and RHS of the batching equation be in the same cyclic subgroup of $\mathbb{Z}_p^*$.

Theorem 2. *Suppose p and q are large primes, such that $p = 2q+1$. G, of order q and generator g, is a cyclic multiplicative subgroup in $\mathbb{Z}_p^*$. For $j = 1, \ldots, m$ and $x \in_R \mathbb{Z}_q^*$, $y = g^x, z_j \in \mathbb{Z}_p^*$, l is a security parameter satisfying $2^l < q$ and $t_j \in_R \{1, \ldots, 2^l\}$. If $\exists k \in \{1, \ldots, m\} \wedge \log_g y \neq \log_{c_k} \pm z_k \bmod p$, then $\log_g y \neq \log_{\prod_{j=1}^m c_j^{t_j}} \prod_{j=1}^m z_j^{t_j}$ with a probability of no less than $1 - 2^{-l}$.*

Due to space restrictions and similarity of Theorem 2 and Theorem 1, we defer the proof for Theorem 2 to the full version of the paper.

3.3 Screening

For m ciphertexts processed in threshold decryption, the previous two theorems are suited to batch each verification of valid decryption shares produced by one participant P_i. Thus, if the batch verification fails, we can identify that particular participant to be dishonest. This is examined in detail in Section 4 and Section 5.

In this subsection, we briefly explain another type of batch verifying valid decryption shares using a common base (same ciphertexts, different participants). If there is only one message in the threshold decryption process ($m = 1$), we can slightly modify the two theorems above to verify valid decryption shares produced by all the participants $\{P_i\}$ together as:

$$\log_g(\prod_{i=1}^{n} y_i^{t_i}) \stackrel{?}{=} \log_c(\prod_{i=1}^{n} z_i^{t_i})$$

We call this technique 'screening' because it can only detect invalid decryption share(s), but is unable to identify the dishonest participant(s). However, divide and conquer, cut and choose, or binary search method [13] can be applied for identifying the bad decryption share(s), thus identifying the dishonest participant(s). Note that this technique only offers considerable performance increase if used in identifying dishonest participants in a large group (i.e: n is large).

4 Applications in Threshold Decryption

In this section, we present the application of our batching theorems (Section 3) to batch verify threshold versions of two popular cryptosystems - threshold ElGamal and RSA. We apply Theorem 2 to batch verify threshold ElGamal, and

Theorem 1 to batch verify threshold RSA. The protocols presented in this section are based on Chaum-Pedersen [6] with a slight modification where the verifier randomly selects the small exponents on the first step.

4.1 Batch Verification in ElGamal

Theorem 2 is suitable to batch verify threshold ElGamal as:

1. For threshold version of ElGamal, the group G is the subgroup of $\mathbb{Z}_p^*$ with an order q.
2. For $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, $v_i \in \mathbb{Z}_p^*$ and $z_{i,j} \in \mathbb{Z}_p^*$ can easily be checked by testing whether $0 < v_i, z_{i,j} < p$.
3. The values $g \in G$ and $\alpha_j \in G$ are publicly verifiable by testing $\left(\frac{g}{p}\right) = 1$ and $\left(\frac{\alpha_j}{p}\right) = 1$ (using the Legendre symbol as in [12]). This proves g and α_j to be generators of G, if $g \neq 1$.
4. For $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, $t_{i,j}$ can be chosen randomly while still satisfying $t_{i,j} < 2^l < q$.

According to Theorem 2, verification of PEQDL in threshold ElGamal (Equation 1) can be batched using SE test as:

$$\log_g(v_i) = \log_{\prod_{j=1}^m \alpha_j^{t_{i,j}}} \left(\prod_{j=1}^m z_{i,j}^{t_{i,j}}\right) \tag{3}$$

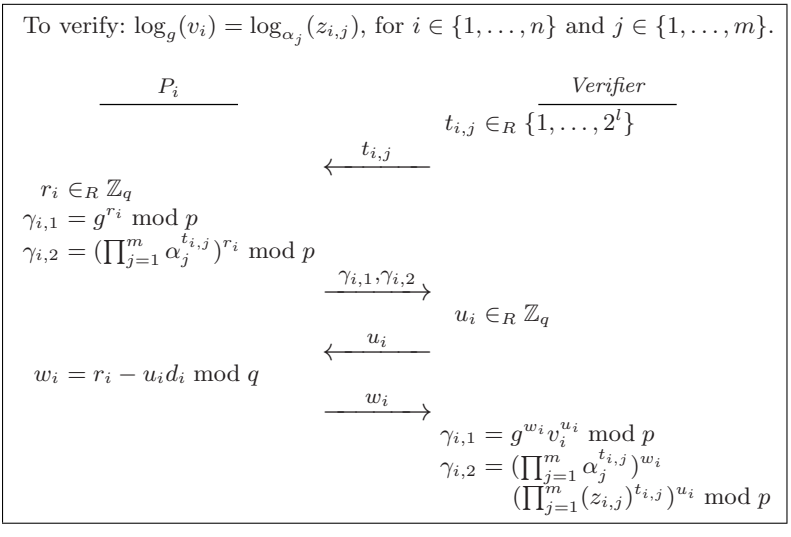

Fig. 2. Batch verification of valid decryption shares for threshold version of ElGamal cryptosystem.

Interactive batch verification protocol for threshold version of ElGamal is shown in Figure 2. Using a hash function and employing the well-known Fiat-Shamir heuristic [10], the protocol can be made non-interactive by producing the challenge u_i using a collision-resistant hash function H, where $H : (0,1)^* \rightarrow \mathbb{Z}_q$ and $j \in \{1, \ldots, m\}$, as follows:

$$u_i = H(\gamma_{i,1}, \gamma_{i,2}, g, v_i, \alpha_j, z_{i,j})$$

Producing the small exponents non-interactively requires a different scenario further explained in Section 5.2. We slightly extend the coin-flipping protocol for the participants to provide a shared source of randomness. This is required in order to prevent a prover from cheating by trying multiple $z_{i,j}$ values until a suitable $t_{i,j}$ value is found. The random values provided are then used to compute the small exponents using a collision-resistant hash function. These are conducted during the shared decryption stage. The protocol to produce the small exponents is shown in Figure 3 and is detailed as below.

1. Each participant (prover) P_i selects a random value τ_i, commits to it using a suitable commitment function, e.g: a hash function as $H(\tau_i)$, and publishes the commitment.
2. Each participant P_i then produces and publishes their decryption share as $z_{i,j} = \alpha_j^{d_i}$.
3. The random value τ_i selected in the first step is then revealed by publishing it.
4. The random small exponents are then calculated using a collision-resistant hash function as: $t_j = H(\tau_i, \alpha_j, j)$, where $i = \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$.

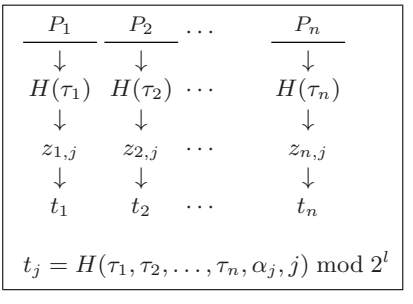

Fig. 3. Producing the small exponents non-interactively

Note that the use of digital signature on the published values is required to authenticate them. Non-interactively, each prover uses the same small exponents t_j as opposed to using different $t_{i,j}$ values provided by the verifier for each prover in the interactive version.

The prover then publishes $(\gamma_{i,1}, \gamma_{i,2}, w_i)$ for public verification. The verification process can be conducted publicly by calculating the small exponents and challenge as above, and checking:

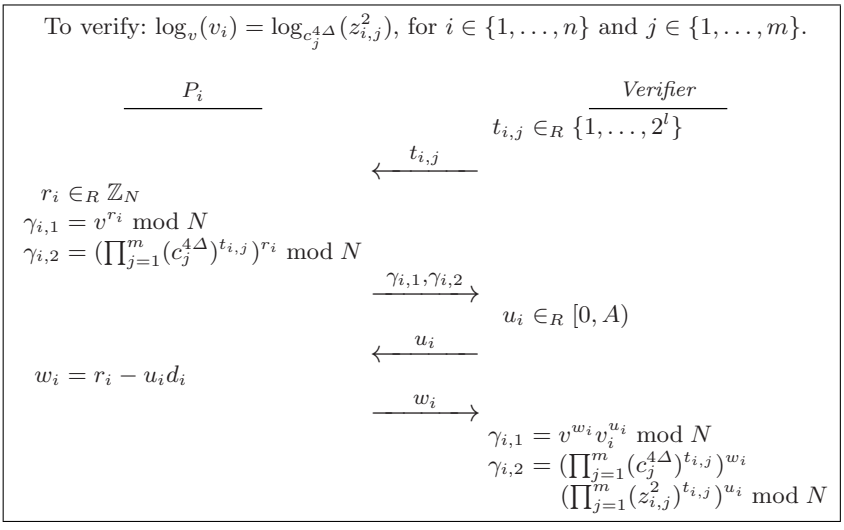

Fig. 4. Batch verification of valid decryption shares for threshold version of RSA cryptosystem.

$$\gamma_{i,1} = g^{w_i} v_i^{u_i} \bmod N$$
$$\gamma_{i,2} = (\prod_{j=1}^{m} (\alpha^{t_j})^{w_i} (\prod_{j=1}^{m} (z_{i,j})^{t_j})^{u_i} \bmod N$$

If all these are satisfied, the verification is accepted. Otherwise, it fails.

We are only convinced that if there exists k where $1 \leq k \leq m$ and $\log_g(v_i^2) = \log_{\alpha_j}(z_{i,j}^2)$, the batch verification can only be passed with negligible probability. Namely, unless $z_{i,j} = \pm\alpha_j^{d_i}$, the batch verification will always fail. Thus, our batch verification result is not yet satisfactory as $s_j = -\alpha_j^{d_i}$ may also satisfy our batch verification. This will lead to incorrect decryption. To fix this, the decryption requires one extra step, i.e: multiplying s_j with (-1) when $s_j \notin G$. After s_j is recovered through the threshold decryption procedure, we test if $\left(\frac{s_j}{p}\right) = 1$ (using the Legendre symbol). If it is accepted, $s_j \in G$. Otherwise, $s_j = -s_j \bmod p$. Then the original secret message is recovered as $\frac{\beta_j}{s_j} \bmod p$. The additional cost is only one exponentiation.

4.2 Batch Verification in RSA

Theorem 1 is applicable to batch the verification of RSA threshold decryption shares as:

1. For threshold version of RSA cryptosystem, G is the cyclic group containing all the squares in $\mathbb{Z}_N^*$ with order $M = p'q'$, the smallest factor of which is $min(p', q')$.

2. The value v is trusted to be a generator of squares in $\mathbb{Z}_N^*$. As v_i is produced using v, and $z_{i,j}^2$ are squares that can be generated by the verifier, thus $v_i, z_{i,j}^2 \in G$ (cyclic subgroup of squares in $\mathbb{Z}_N^*$).
3. The value of $c_j^{4\Delta}$ is a square, and v is trusted to be squares in $\mathbb{Z}_N^*$ chosen by the trusted dealer. Therefore, both $c_j^{4\Delta}, v \in G$. Thus, $c_j^{4\Delta}$ and v are generators of G (see [18]) with a very large probability $(1 - \frac{p'+q'-1}{p'q'})$.
4. For $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, $t_{i,j}$ can be chosen randomly while still satisfying $t_{i,j} < 2^l < min(p', q')$.

According to Theorem 1, SE test can be implemented as the following:

$$\log_v(v_i) = \log_{\prod_{j=1}^{m}(c_j^{4\Delta})^{t_{i,j}}}(\prod_{j=1}^{m}(z_{i,j}^2)^{t_{i,j}}) \tag{4}$$

Interactive batch verification for threshold version of RSA is shown in Figure 4. Where $A \times ord(G)$ is much smaller than N, the challenge u_i must be chosen in $[0, A)$ such that the shared secret key d_i is statistically hidden in the response w_i as in [15,2]. Analysis in [15] suggests the minimum size of the challenge $|A|$ to be 80 bits, and 128 bits for more secure applications.

Using a hash function and employing the well-known Fiat-Shamir heuristic, the protocol is made non-interactive similar to the previous section. The prover produces the small exponents as shown in the previous section (Figure 3), and produces the challenge u_i using a collision-resistant hash function H, where $H : (0,1)^* \mapsto \mathbb{Z}_M$ and $j \in \{1, \ldots, m\}$, similar to the previous section as follows:

$$t_j = H(t_1, t_2, \ldots, t_n, \alpha_j, j) \bmod 2^l$$
$$u_i = H(\gamma_{i,1}, \gamma_{i,2}, v, v_i, c_j^{4\Delta}, z_{i,j}^2)$$

The prover then publishes $(\gamma_{i,1}, \gamma_{i,2}, w_i)$ for public verification. The verification process can be conducted publicly by calculating the small exponents and challenge as above, and checking:

$$\gamma_{i,1} = v^{w_i} v_i^{u_i} \bmod N$$
$$\gamma_{i,2} = (\prod_{j=1}^{m}(c_j^{4\Delta})^{t_j})^{w_i}(\prod_{j=1}^{m}(z_{i,j}^2)^{t_j})^{u_i} \bmod N$$

If all these are satisfied, the verification is accepted. Otherwise, it fails.

Unlike in threshold ElGamal, extra verification to ensure that decryption shares passing the batch verification are not $-z_{i,j}$ is not necessary. This is because decryption shares $z_{i,j}$ are explicitly squared in the share combining phase to reconstruct the secret message.

5 Security Analysis

5.1 Completeness

Completeness of each of the two protocols in Section 4 is straight-forward. This is because if the batch verification equations in the two protocols are correct, they output positive results.

5.2 Soundness

The two protocols in Section 4 are very similar. They are based on Chaum-Pedersen's protocol. We slightly modify the protocol where the verifier randomly selects the small exponents at the beginning of the protocol run. The proof of soundness for the protocols follows from Chaum-Pedersen's scheme as they are essentially the same. The small exponents $t_{i,j}$ are chosen randomly in a very similar manner ($t_{i,j} < 2^l$) to choosing the random challenge.

Given the same random small exponents and commitments, no matter which challenge is chosen, the prover reveals no other information than the fact that the discrete logarithms of the verification key to the base of verification base equals the discrete logarithms of the product of the decryption shares to the base of the product of the ciphertexts (Equation 3 and 4).

In the interactive version, the probability for a prover to cheat is negligible. It is not feasible to forge the decryption shares where the verification is accepted without the knowledge of the share decryption key. Also, where the prover indeed holds the decryption key share, the probability of producing bad decryption shares where the verification is accepted is also negligible. This is because the small exponents and challenge are chosen randomly by the verifier. For example, in batching the verification of correct ElGamal decryption shares, the probability of a prover guessing a correct random small exponent and challenge, and the verification is accepted is $2^{-l}q^{-1}$.

In the non-interactive version, we also follow Chaum-Pedersen's protocol with a slight addition in choosing the random small exponents (Figure 3) based on the coin-flipping protocol. We avoid the use of a hash function with the input (the decryption share $z_{i,j}$) chosen by a single prover to compute the small exponents. This is because it might be possible for a dishonest participant to try fixing the decryption share(s) and produce the small exponents, such that the verification is accepted and the share combining fails. A distributed source of randomness (based on the coin-flipping protocol) is required as the small exponents are only of length l, where l is small.

The probability of a prover forging his decryption share and fixing the small exponent share is negligible. This is because the prover is required to commit to the random share first before publishing his decryption share, and the small exponents are produced by hashing the combined random shares (common reference string) of all the participants. As a collision-resistant hash function is used to produce the small exponents, a prover can only attempt to forge his decryption share if all the participants collude.

The rest of the protocol is a Σ-protocol [7], and thus has a special soundness property as proven in [7]. The proof of soundness for the batching operation has been proven in Section 3.1 and Section 3.2.

5.3 Error Probability

In any of the two batch verification protocols presented, the probability that a dishonest participant is discovered is overwhelmingly large as the following:

- As indicated by Theorem 1, the probability that the batch verification equation is satisfied given incorrect share decryption(s) is 2^{-l}.
- As the prover has to guess the challenge u_i at random, the probability that the batch verification test is accepted where the batch verification equation is not satisfied is $\frac{1}{q}$.
- Therefore, the probability that the batch verification is not accepted given incorrect share decryption is $(1 - 2^l)(1 - q^{-1})$

As q^{-1} is very small, e.g: 2^{-1024}, the probability that a dishonest participant being undetected given incorrect share decryption(s) is approximately 2^{-l}.

6 Efficiency Analysis

Most schemes employing threshold decryption take the decryption process for granted. For example, in the mixnet scheme by Boneh and Golle [4], they focus on improving the efficiency of correct mixing operation and only mention the use of threshold decryption. Using our result, the overall performance of the mixnet scheme can be greatly improved.

We follow Bellare *et al.* in measuring the cost of our algorithms, where $ExpCost^m(l)$ denotes the time to compute m exponentiations in a common base with different exponents of the same length l. The computational cost comparison of naive verification against interactive batch verification for threshold versions of two popular cryptosystems - ElGamal and RSA - is summarised in Table 1 in terms of the number of modular multiplications required.

Suppose $ExpCost(x) = 1.5x$ and $ExpCost^y(x) = y + 0.5xy$. Table 1 also illustrates an example of verifying valid decryption shares from 50 ($m = 50$) ciphertexts for 10 participants ($n = 10$, $\log_2 \Delta \approx 22$), where the length of the integers involved is 1024 bits and the acceptable error is 2^{-20} ($l = 20$). Implemented in the mixnet of Boneh and Golle, our result offers a great reduction of the computational cost in the threshold decryption phase of the shuffled ciphertexts to be decrypted in the final phase of mixnet.

The performance increase in Table 1 is calculated based on the difference of modular multiplication required in the naive and batch version. According to Table 1, it is estimated that performance increase when batch verification is employed would be about 97%.

Our results offer better proving and verification performance, while the probability of an invalid decryption share being accepted is no more than 2^{-l}. When m increases, the computational verification cost saved by using our scheme also increases.

7 Conclusion

The SE test by Bellare *et al.* is originally designed to batch verify modular exponentiations in the context of signature verification. We modified and extended the scheme to batch verify PEQDL in the context of threshold decryption.

Table 1. Performance (number of modular multiplications required) comparison on interactive batch verification of valid decryption shares for threshold versions of two popular cryptosystems, with 10 participants (decrypting authorities, $n = 10$), 50 secret messages to process ($m = 50$), and 2^{-20} acceptable error ($l = 20$).

		Naive	Batch
ElGamal	Each prover	$m(2ExpCost(\log_2 q) + 1)$	$ExpCost^m(\log_2 l)$
			$+2ExpCost(\log_2 q) + 1$
		$= 153650$	$= 3623$ (97.64% more efficient)
	Verifier	$2nmExpCost^2(\log_2 q)$	$n(2ExpCost^m(\log_2 l)$
			$+2ExpCost^2(\log_2 q))$
		$= 1026000$	$= 31520$ (96.93% more efficient)
RSA	Each prover	$m(2ExpCost(\log_2 M) + 1)$	$ExpCost^m(\log_2 \Delta + \log_2 l + 2)$
			$+2ExpCost(\log_2 M) + m + 2$
		$= 153650$	$= 4274$ (97.22% more efficient)
	Verifier	$2nmExpCost^2(\log_2 M)$	$n(2ExpCost^m(\log_2 \Delta + \log_2 l + 2)$
			$+2ExpCost^2(\log_2 M))$
		$= 1026000$	$= 43520$ (95.76% more efficient)

The scheme presented in this paper greatly improves the efficiency of identifying correct decryption shares (honest participants) with an overwhelmingly high probability when a large number of ciphertexts are involved. The bucket test by Bellare *et al.* (a variant of SE test) can similarly be modified and extended to achieve better efficiency.

It is quite straight-forward to apply the scheme to batch verify decryption shares in threshold version of Paillier cryptosystem [9], similar to that of threshold version of RSA. Due to space constraints, we provide the application in the full version of this paper.

Our scheme can easily be implemented in cryptographic applications employing threshold decryption in lowering their computational cost. This offers great performance benefit to various applications requiring verification of many PEQDLs, such as in secure e-auction or e-voting schemes.

Acknowledgements. We acknowledge the support of the Australian government through ARC Discovery 2002, Grant No: DP0211390; ARC Discovery 2003, Grant No: DP0345458; and ARC Linkage International fellowship 2003, Grant No: LX0346868.

References

1. Masayuki Abe and Koutarou Suzuki. M+1-st price auction using homomorphic encryption. In *Public Key Cryptography—PKC 02*, pages 115–124, 2002.
2. Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Twentieth Annual ACM Symposium on Principles of Distributed Computing*, pages 274–283, 2001.

3. Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology—EUROCRYPT 98*, pages 236–250, 1998.
4. Dan Boneh and Philipe Golle. Almost entirely correct mixing with applications to voting. In *ACM Conference on Computer and Communications Security—CCS 02*, pages 68–77, 2002.
5. Colin Boyd and Chris Pavlovski. Attacking and repairing batch verification schemes. In *Advances in Cryptology—ASIACRYPT 00*, pages 58–71, 2000.
6. David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Advances in Cryptology—CRYPTO 92*, pages 89–105, 1993.
7. Ronald Cramer and Ivan Damgård. Secure signature schemes based on interactive protocols. In *Advances in Cryptology—CRYPTO 95*, pages 297–310, 1995.
8. Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology—EUROCRYPT 97*, pages 103–118, 1997.
9. Ivan Damgaård and Mats Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography—PKC 01*, pages 119–136, 2001.
10. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO 86*, pages 186–194, 1986.
11. Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Financial Cryptology—FC 00*, pages 90–104, 2000.
12. Fumitaka Hoshino, Masayuki Abe, and Tetsutaro Kobayashi. Lenient/strict batch verification in several groups. In *Information Security, 4th International Conference, ISC 01*, pages 81–94, 2001.
13. Jaroslaw Pastuszak, Dariusz Michatek, Josef Pieprzyk, and Jennifer Seberry. Identification of bad signatures in batches. In *Public Key Cryptography—PKC 00*, pages 28–45, 2000.
14. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—CRYPTO 91*, pages 129–140, 1992.
15. Guillaume Poupard and Jacques Stern. On the fly signatures based on factoring. In *ACM Conference on Computer and Communications Security—CCS 99*, pages 37–45, 1999.
16. Kazue Sako. An auction protocol which hides bids of losers. In *Public Key Cryptography—PKC 00*, pages 422–432, 2000.
17. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.
18. Victor Shoup. Practical threshold signatures. In *Advances in Cryptology—EUROCRYPT 00*, pages 207–220, 2000.