Improved Methods to Perform Threshold RSA

Brian King

University of Wisconsin-Milwaukee, Milwaukee,WI 53201 USA
Motorola Labs, Schaumburg IL 60010, USA
brking@execpc.com

Abstract. A t out of n threshold scheme is such that shares are distributed to n participants so that any set of t participants can compute the secret, whereas any set of less than t participants gain no information about the secret. In [4], Desmedt and Frankel introduced a threshold scheme that can be used with any finite Abelian group. Hence it can be used to provide threshold RSA. In this scheme, the size of the share is on the order n times the size of the secret. Further, due to a complicated algebraic setting, and the large shares, this schemes requires a "large" amount of computations. Recent work have addressed how to reduce the resource requirements. Within this paper we provide improved methods and demonstrate the computational requirements of the Desmedt-Frankel scheme using our method is, in many cases, better than other existing threshold RSA signature schemes.

Keywords: threshold secret sharing, threshold cryptography, threshold RSA, cyclotomic polynomials

1 Introduction

RSA [18] is an important cryptographic scheme. The development of threshold RSA was problematic due to the fact that the modulus $\phi(N)$, [1] [2] as well any multiple, cannot be leaked to any of the shareholders. Threshold RSA has been examined in [10,5,6], then in [13,4,3], and most recently in [11,12,17,20]. The Desmedt-Frankel scheme [4] was the first secure threshold RSA sharing scheme. This is a zero-knowledge threshold scheme. Further this scheme is a group independent scheme. That is, the shareholder reconstruction of the secret key is independent of the group.[3] Group independent schemes provide a flexible threshold secret sharing. However, there is a disadvantage when using this scheme. The disadvantage of the scheme is the amount of resources it requires, in the sense of memory (share size) and processing (computational time). The memory requirement is caused by share expansion. That is, the share expansion is such

[1] Here $N = p_1 p_2$, the product of two distinct primes, and $\phi(N) = (p_1 - 1) \cdot (p_2 - 1)$.
[2] The true modulus is the Carmichael function $\lambda(N)$, which is a divisor of $\phi(N)$
[3] In most applications of threshold cryptography, the reconstructed value is a function of the secret key, not the secret key. For example a signature. In such cases the shareholder will obviously have to perform algebraic operations within the ring $\mathbf{Z}_N$.

T. Okamoto (Ed.): ASIACRYPT 2000, LNCS 1976, pp. 359–372, 2000.
© Springer-Verlag Berlin Heidelberg 2000

that shares will consist of $O(n)$ subshares drawn from the keyspace. The processing cost comes from the computing requirements. Computations will need to be performed on these large shares. Moreover, computations will need to be performed in what appears to be a complicated algebraic structure. These resource requirements (and interest in development of robust, proactive, and/or verifiable threshold RSA) have led to developing other schemes. However the algebraic structure of the Desmedt-Frankel scheme is used in some of these other schemes (for example [3,13]). Work has been initiated in relieving some of the computational requirements for the Desmedt-Frankel scheme. In [8], the authors established that within the Desmedt-Frankel scheme, the share size for each participant could be halved. In [15], we showed how to speed up the required computations when using the Desmedt-Frankel scheme to achieve a threshold RSA signature scheme. Most of the computation improvements were developed for the shareholder. Further, we provided a computation comparison of the Desmedt-Frankel signature scheme with the signature scheme recently developed by Shoup in [20]. Using this comparison we pointed out that in many cases the Desmedt-Frankel scheme performed better than the signature scheme developed by Shoup. Here we will provide further improvement in the performance of the Desmedt-Frankel scheme. Our analysis will show that the computational requirements on the shareholder, as described here, is always less than or equal to the computational requirement for the shareholder as described in [15]. Furthermore, we will show that the computational requirement for the distributor is significantly better than those computation requirements stated by both [4] and [15].

1.1 The Desmedt Frankel Scheme

$\mathcal{K}$ represents a finite abelian group. The secret k is selected from $\mathcal{K}$. A prime q is chosen such that $q \geq n+1$. (We can assume, due to Bertrand's postulate [16], that $O(q) = O(n)$.) Let u represent a root of the cyclotomic polynomial $m(x) = \sum_{j=0}^{q-1} x^j$. Many of the computations are performed in the ring $\mathbf{Z}[u] \cong \mathbf{Z}[x]/m(x)$. Notice that $\alpha_i = \sum_{j=0}^{i-1} u^j$ is a unit (an invertible element in $\mathbf{Z}[u]$) for each i $(1 \leq i \leq q-1)$ and that $\alpha_i - \alpha_j$ is a unit for all distinct i, j, with $1 \leq i, j \leq q-1$.

Consider the group $\mathcal{K}^{q-1}$ where $\mathcal{K}^{q-1} = \mathcal{K} \times \mathcal{K} \times \cdots \times \mathcal{K}$. If $\boldsymbol{x} \in \mathcal{K}^{q-1}$ then $\boldsymbol{x} = [x_0, x_1, \ldots, x_{q-2}]$. For all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{K}^{q-1}$

$$\boldsymbol{x}_1 + \boldsymbol{x}_2 = [x_{1,0} + x_{2,0}, x_{1,1} + x_{2,1}, \ldots, x_{1,q-2} + x_{2,q-2}].$$

Let $\mathbf{0} = [0, 0, \ldots, 0]$, where 0 denotes the identity in $\mathcal{K}$. For all $b \in \mathbf{Z}$, $b\boldsymbol{k} = [bk_0, bk_1, \ldots, bk_{q-2}]$. where bk_i represents the element in $\mathcal{K}$, formed by applying k_i to itself b times. For $u \in \mathbf{Z}[u]$, $u\boldsymbol{k} = [0, k_0, k_1, \ldots, k_{q-3}] + [-k_{q-2}, \ldots, -k_{q-2}] = [-k_{q-2}, -k_{q-2}+k_0, \ldots, -k_{q-2}+k_{q-3}]$. Then $u^{i+1}\boldsymbol{k} = u(u^i\boldsymbol{k})$. For all polynomials f in u with integer coefficients, $f(u) = b_0 + b_1 u + \cdots + b_k u^k$, $f(u)$ is defined by $f(u)\boldsymbol{k} = \sum_{i=0}^{k} b_i(u^i\boldsymbol{k})$. Then $\mathcal{K}^{q-1}$ is a module over $\mathbf{Z}[u]$ (for more information see [1]).

1.2 How Shares Are Computed

Given secret k, we represent the secret by $\boldsymbol{k} = [k, 0, \ldots, 0] \in \mathcal{K}^{q-1}$. There are two alternative to generate shares.

Each shareholder P_i $(i = 1, \ldots, n)$ is given share $\boldsymbol{s}_i$ in the following manner: First, $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{t-1}$ are chosen uniformly random from $\mathcal{K}^{q-1}$. For $i \geq t$, $\boldsymbol{s}_i$ is defined by:

$$\boldsymbol{s}_i = y_{i,C_i}^{-1} \cdot \left(\boldsymbol{k} - \sum_{\substack{j \neq i \\ j \in C_i}} y_{j,C_i} \cdot \boldsymbol{s}_j \right) \tag{1}$$

where $C_i = \{1, 2, \ldots, t-1, i\}$ and for each $j \in C_i$

$$y_{j,C_i} = \frac{\displaystyle\prod_{\substack{h \in C_i \\ h \neq j}} (0 - \alpha_h)}{\displaystyle\prod_{\substack{h \in C_i \\ h \neq j}} (\alpha_j - \alpha_h)} \tag{2}$$

An alternate manner to compute the shares is to choose $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{t-1}$ uniformly random from $\mathcal{K}^{q-1}$, such that share $\boldsymbol{s}_i$ is determined by

$$\begin{bmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \vdots \\ \boldsymbol{s}_l \end{bmatrix} = \begin{bmatrix} 1 & \alpha_1 & \cdots & \alpha_1^{t-1} \\ 1 & \alpha_2 & \cdots & \alpha_2^{t-1} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha_l & \cdots & \alpha_l^{t-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{k} \\ \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_{t-1} \end{bmatrix} \tag{3}$$

Therefore for each i, $\boldsymbol{s}_i = \boldsymbol{k} + \alpha_i \cdot \boldsymbol{c}_1 + \cdots + \alpha_i^{t-1} \boldsymbol{c}_{t-1}$. Hence $\boldsymbol{s}_i = g(\alpha_i)$ where $g(x) = \boldsymbol{k} + x \cdot \boldsymbol{c}_1 + \cdots + x^{t-1} \boldsymbol{c}_{t-1}$. (In this case it is possible to compute the shares using Horner's algorithm.)

1.3 How the Secret k Is Computed

When a set B of t participants wish to compute $k \in \mathcal{K}$, they can determine $\boldsymbol{k}$, of which the first component of $\boldsymbol{k}$, is the secret k. The participants determine $\boldsymbol{k}$ by

$$\boldsymbol{k} = \sum_{i \in B} y_{i,B} \cdot \boldsymbol{s}_i \tag{4}$$

where $y_{i,B}$ is defined by (2). We will use the F_0 to denote the function which maps any $(q-1)$ tuple to its first coordinate. Thus $k = F_0(\boldsymbol{k})$.

1.4 How Much Time Is Needed to Perform the Necessary Algebraic Operations

One of our concerns is the amount of time each shareholder uses to perform algebraic operations, another is the amount of time the distributor needs to

perform algebraic operations. As described by equations (1) and (4), a required computation appears to be $y_{i,B}$. (Where in the case of distributor, they need to calculate such a $y_{i,B}$ "$t(n-t)$" times.)

There are two approaches to perform the needed calculation. The first approach is described as follows: to perform $y_{i,B} \cdot \boldsymbol{s}_i$, perform

$$y_{i,B} \cdot \boldsymbol{s}_i = \prod_{\substack{h \in C_i \\ h \neq j}} (0 - \alpha_h) \left(\prod_{\substack{h \in C_i \\ h \neq j}} \left(\frac{1}{\alpha_j - \alpha_h} \cdot \boldsymbol{s}_i \right) \right).$$

The running time to compute $y_{i,B} \cdot \boldsymbol{s}_i$ in this manner, as stated by [4] is:

Theorem 1. *[4] The shareholder performs $O(tn^2)$ group operations, and $O(n)$ inverses. In addition to the time to choose randomly $t-1$ shares. The distributor performs $O(t^2n^2(n-t))$ group operations and $O(tn(n-t))$ inverse operations.*

A different method is suggested if group operation is slower than integer multiplication. Instead of performing a series of group operations, a series of $\mathbf{Z}[u]$ operations are performed, until $y_{i,B}$ is formed, then one group operation is performed. In [4], the authors established the following:

Theorem 2. *[4] Each shareholder performs $O(nt \log n)$ group operations, $O(n)$ inverses, and $O(t^3n^2(\log n)^2)$ elementary integer operations. In addition to the time needed to choose $t-1$ random shares, the distributor performs $O(t^2n(n-t)\log_2 n)$ group operations, $O(tn(n-t))$ inverse operations, and $O(t^4n^2(n-t)(\log n)^2)$ elementary integer operations.*

2 Algorithms Which Improved Performance in the Desmedt-Frankel Scheme

In [15], it was illustrated how to improve the performance of the Desmedt-Frankel scheme. Here, a series of results and algorithms were introduced in order to compute more efficiently. We discuss now those results that we will utilize. We will let Γ_0 denote all sets of t participants. (In order to avoid confusion between an inverse of an α_x with an inverse of an $x \in \mathbf{Z}_q^*$, we will represent an inverse of $x \in \mathbf{Z}_q^*$ by r_x.)

Theorem 3. *[15]*
(1) For all i, j, with $i \neq j$, $\alpha_i - \alpha_j = u^j \alpha_{i-j}$.
(2) For all $x, a \in \mathbf{Z}_q^*$, $\dfrac{\alpha_x}{\alpha_a} = \dfrac{\alpha_{ka}}{\alpha_a} = 1 + u^a + \cdots + u^{a(k-1)}$, where $k = xr_a \bmod q$.
(3) If f is the product of r many cyclotomic polynomials of the form $\frac{\alpha_x}{\alpha_a}$, then all coefficients of f are bounded by $-q^{r-1}$ and q^{r-1}.
(4) For all i, $\alpha_x^{-1} = \dfrac{\alpha_{kx}}{\alpha_x}$ where $k = r_x \bmod q$.
(5) For all i and $B \in \Gamma_0$, $y_{i,B} = u^{-(t-1)i} \displaystyle\prod_{\substack{h \in B \\ h \neq i}} \frac{\alpha_h}{\alpha_{h-i}}$.

(6) There exists an algorithm that will calculate the product of r many cyclotomic polynomials of the form $\frac{\alpha_{qx}}{\alpha_a}$ with a running time of $O(rn\log_2 n)$.
(7) There exists an algorithm that will calculate α_x^{-1} with a running time of $O(n\log_2 n)$.
(8) There exist an algorithm that will calculate $y_{i,B}$ with a running time $O(t^2 n\log_2 n)$

As discussed in [15], consider the Desmedt-Frankel scheme implementing a threshold RSA signature. N is the product of two distinct primes, and $\phi(N)$ is the Euler totient function. Then $\mathcal{K}$ is $\mathbf{Z}_{\phi(N)}$. In threshold RSA no participant can be given any information concerning $\phi(N)$. Shares are actually $(q-1)$ dimensional vectors in the module $\mathbf{Z}_{\phi(N)}^{q-1}$. From a shareholder's view this will look like a $(q-1)$ dimensional integer vector. Thus computations can be performed using integer addition. The secret is d, where the RSA public key is (e, N) and $ed = 1 \pmod{\phi(N)}$. Therefore $\boldsymbol{k} = [d, 0, \ldots, 0] = \sum_{i\in B} y_{i,B}\boldsymbol{s}_i$. If a set of t participants would like to sign a message m, then they will not send their subshares of d (i.e. they will NOT send $y_{i,B}\cdot \boldsymbol{s}_i$), but rather they will send partial signatures. They could send $m^{y_{i,B}\cdot \boldsymbol{s}_i}$. If all t participants sent $m^{y_{i,B}\cdot \boldsymbol{s}_i}$ then a combiner would get m^d by

$$m^d = F_0(m^{[d,0,\ldots,0]}) = F_0(\prod_{i\in B} m^{y_{i,B}\cdot \boldsymbol{s}_i}).$$

However we must point out that this method wastes resources. That is, the combiner is actually computing $m^{[d,0,\ldots,0]} = [m^d, 1, 1, \ldots, 1]$, whereas the only element of interest is m^d. Consider the following method of computation. Recall that F_0 is a function which maps a j-tuple to its first coordinate. Now, suppose that B is a set of participants, with $|B| = t$,, and that for all $j \in B$, $\boldsymbol{z}_j \in \mathcal{K}^{q-1}$, $\boldsymbol{z}_j = [z_{j,0}, z_{j,1}, \ldots, z_{j,q-2}]$. Then $F_0(\prod_{j\in B} m^{\boldsymbol{z}_j}) = F_0(m^{\sum z_{j,0}}, m^{\sum z_{j,1}}, \ldots, m^{\sum z_{j,q-2}}) = \prod_{j\in B} F_0(m^{\boldsymbol{z}_j}) = m^{F_0(\sum \boldsymbol{z}_j)} = m^{\sum z_{j,0}}$. So we see that to compute m^d, all that is needed is $F_0(y_{i,B}\cdot \boldsymbol{s}_i)$.

Using a result developed by Desmedt-Frankel in [10], [15] established the following.

Theorem 4. *[15] To compute a partial signature, a shareholder must compute an integer then exponentiate m to this integer. The amount of time required for shareholder computations can be expressed as: The time to compute the integer is $O(nt(t\log_2 n + \log_2 n \log_2 \phi(N))$. The time to exponentiate is $O((t\log_2 n + \log_2 \phi(N))(\log_2 N)^2)$.*

Theorem 5. *[15] The distributor is required to determine $O((t-1)n)$ elements from $\mathbf{Z}_{\phi(N)}$. The distributor needs to compute $O((n-t+1)n)$ elements from $\mathbf{Z}_{\phi(N)}$. The amount of time required for the distributor computations is $O(nt^2(n-t)\log_2 n \log_2 \phi(N)))$.*

Both Theorem 4 and Theorem 5 incorporate a technique which calls for the computation of $y_{i,B}$ and then applying $y_{i,B}\cdot \boldsymbol{s}$ (a technique proposed by Theorem 2). We now provide a better alternative. This method will apply a technique very similar to the technique proposed by Theorem 1.

3 Applying a Cyclotomic Polynomial to a Share

Given a share $\boldsymbol{s} = [s_0, s_1, \ldots, s_{q-2}]$, we will refer to s_0 as the *zeroth* term, s_1 as the first term, etc. Recall the application of u to a share $\boldsymbol{s}$ as: $u\boldsymbol{s} = [-s_{q-2}, s_0 - s_{q-2}, s_1 - s_{q-2}, \ldots, s_{q-3} - s_{q-2}]$. Then $u^2\boldsymbol{s} = [s_{q-2} - s_{q-3}, -s_{q-3}, s_0 - s_{q-3}, \ldots, s_{q-4} - s_{q-3}]$, $u^3\boldsymbol{s} = [s_{q-3} - s_{q-4}, s_{q-2} - s_{q-4}, \ldots, s_{q-5} - s_{q-4}]$, and so forth. In general

$$u^a\boldsymbol{s} = [s_{q-a} - s_{q-a-1}, s_{q-a+1} - s_{q-a-1}, s_{q-a+2} - s_{q-a-1} \ldots, s_{q-a-2} - s_{q-a-1}] \quad (5)$$

Where all but one of the terms is a difference of two subshares, this term which is not the difference of two subshares is the $a-1^{st}$ term and is the negative of the subshare s_{q-a-1}. Further observe that when considering all positive terms of the difference each one of the original subshares occurs once, except for the subshare s_{q-a-1} (which is the negative term for all the differences). Extend the definitions of the subshares s_i by defining $s_q = s_0$, and for any integer i, $s_i = s_{i \bmod q}$. Due to space, we have omitted many of the proofs to our results.

Theorem 6. *To compute $u^a\boldsymbol{s}_i$ requires $O(n)$ additions and 1 inverse within the group $\mathcal{K}$.*

In an effort to develop symmetry we will define an artificial subshare s_{q-1}. We are only creating this artificial subshare to illustrate a property concerning cyclotomic polynomials applied to $\boldsymbol{s}$. In order for our computations to remain correct there is only one choice for s_{q-1}, we define s_{q-1}=0 for all participants.

Now consider the effect of our definition of s_{q-1}, we have $u^a\boldsymbol{s} = [s_{q-a} - s_{q-a-1}, s_{q-a+1} - s_{q-a-1}, s_{q-a+2} - s_{q-a-1} \ldots, s_{q-a-2} - s_{q-a-1}]$, where the j^{th} term of $u^a\boldsymbol{s}$ is $s_{q-a+j} - s_{q-a-1}$. Then $\alpha_i\boldsymbol{s} = [\sum_{j=0}^{i-1} s_{q-j} - cst_i, \sum_{j=0}^{i-1} s_{q-j+1} - cst_i, \sum_{j=0}^{i-1} s_{q-j+2} - cst_i, \ldots, \sum_{j=0}^{i-1} s_{q-j+q-2} - cst_i]$ where $cst_i = \sum_{j=0}^{i-1} s_{q-1-j}$. (Observe that in our definition of cst_i we have included $i-1$ additions, in reality there are only $i-2$ additions for the case for $j = 0$ recall that $s_{q-1-j} = s_{q-1} = 0$.) If we represent $\alpha_i\boldsymbol{s} = [a'_0, a'_1, \ldots, a'_{q-2}]$, then

$$\begin{aligned} a'_0 &= s_{q-0} + s_{q-1} + \cdots + s_{q-(i-1)} - cst_i, \\ &\vdots \\ a'_j &= s_j + s_{j-1} + s_{j-2} + \cdots + s_{j-(i-1)} - cst_i. \end{aligned}$$

Thus, for $1 \leq j \leq q-2$,

$$a'_j = a'_{j-1} + s_{j-1} - s_{q-(i-j)}. \quad (6)$$

Now observe that $a'_0 = s_q - s_{q-i} = s_0 - s_{q-i}$.

Theorem 7. *To compute $\alpha_i\boldsymbol{s}$ requires $O(n)$ additions and $O(n)$ inverses within the group $\mathcal{K}$.*

We now consider the cost of computing $\frac{\alpha_{ax}}{\alpha_a}\boldsymbol{s}$. To this end, we generalize $u\boldsymbol{s}$ to include the artificial subshare s_{q-1}. So for all $\boldsymbol{s}$ we extend $\boldsymbol{s}$ by

$$\boldsymbol{s} = [s_0, \ldots, s_{q-2}] = [s_0, \ldots, s_{q-2}, 0]_{ext} = [s_0, \ldots, s_{q-2}, s_{q-1}]_{ext} \tag{7}$$

Then for $\boldsymbol{s} = [s_0, \ldots, s_{q-2}, s_{q-1}]_{ext}$, and any $b \in \mathbf{Z}$, $b\boldsymbol{s} = [bs_0, \ldots, bs_{q-2}, bs_{q-1}]_{ext}$. Further, extend $u\boldsymbol{s} = [s_{q-1} - s_{q-2}, s_0 - s_{q-2}, \ldots, s_{q-3} - s_{q-2}, s_{q-2} - s_{q-2}]_{ext}$. Notice that both $b\boldsymbol{s}$ and $u\boldsymbol{s}$ are the same as the true $b\boldsymbol{s}$ and $u\boldsymbol{s}$ with a 0 appended at the end.

Theorem 8. *For all a, with $1 \leq a \leq q-1$, $u^a\boldsymbol{s} = [s_{q-a} - s_{q-a-1}, s_{q-a+1} - s_{q-a-1}, s_{q-a+2} - s_{q-a-1} \ldots, s_{q-a-2} - s_{q-a-1}, s_{q-a-1} - s_{q-a-1}]_{ext}$. Thus the j^{th} term of $u^a\boldsymbol{s}$ is $s_{q-a+j} - s_{q-a-1}$.*

Observe that for each $x \in \{0, \ldots, q-1\}$ there exists an integer $i \in \{0, \ldots, q-1\}$ such that $x = ia \bmod q$. Of course if $x = 0$ then $i = 0$, otherwise $i = xr_a \bmod q$, where we have used r_a to represent the inverse of a in the field $\mathbf{Z}_q$. Next observe from the above theorem that if we represent $u^a\boldsymbol{s}$ by $u^a\boldsymbol{s} = [s'_0, s'_1, \ldots, s'_{q-1}]_{ext}$, then $s'_x = s_{q-a+x} - s_{q-a-1}$. Therefore $s'_{ia} = s_{q-a+ia} - s_{q-a-1}$. Now observe that $q - a = (qa - a) \bmod q = (q-1)a \bmod q$. Let $\epsilon_a \in \mathbf{Z}_q^*$ such that $\epsilon_a a = q - 1 \bmod q$. Then $(\epsilon_a - 1)a = q - a - 1 \bmod q$. Hence $s'_x = s'_{ia} = s_{(q-1)a+ia} - s_{(\epsilon_a-1)a} = s_{(q-1+i)a} - s_{(\epsilon_a-1)a} = s_{(i-1)a} - s_{(\epsilon_a-1)a}$.

Corollary 9. *For all integers j, with $1 \leq j \leq q-1$, $u^{aj}\boldsymbol{s} = [s_0^{(j)}, s_1^{(j)}, \ldots, s_{q-1}^{(j)}]_{ext}$, where $s_{ia}^{(j)} = s_{(i-j)a} - s_{(\epsilon_a - j)a}$.*

Theorem 10. *For all j, with $1 \leq j \leq q-1$, $(1 + u^a + u^{2a} + \cdots + u^{ja})\boldsymbol{s} = [b_0, b_1, \ldots, b_{q-1}]_{ext}$, such that $b_{ia} = s_{ia} + \sum_{x=1}^{j} s_{(i-x)a} - \sum_{x=1}^{j} s_{(\epsilon_a - x)a}$. (Recall that $\epsilon_a = q - 1 \bmod q$.)*

We make the following observations. Consider the b_i from Theorem 10. First $b_0 = s_0 + \sum_{x=1}^{j} s_{(q-x)a} - \sum_{x=1}^{j} s_{(\epsilon_a - x)a}$. Next $b_a = s_a + \sum_{x=1}^{j} s_{(1-x)a} - \sum_{x=1}^{j} s_{(\epsilon_a - x)a} = b_0 + s_a - s_{(-j)a}$. In general

$$b_{(i+1)a} = b_{ia} + s_{(i+1)a} - s_{(i-j)a}. \tag{8}$$

Also observe that for all i,

$$0 \leq |b_{ia}| \leq q \cdot \max_j |s_j|. \tag{9}$$

Theorem 11. *For each a and x, with $1 \leq a, x \leq q-1$ it requires $O(n)$ additions and $O(n)$ inverses to compute $\frac{\alpha_{ax}}{\alpha_a}\boldsymbol{s}$. It also requires $O(n \log_2 n)$ elementary operations. (The elementary operations cost represents the time required to increment ia for $i = 0, \ldots, q-1$.)*

Regarding shareholder computations, we have not referred to the arithmetic operations as "arithmetic operations in the group $\mathcal{K}$". The shareholder may not be given enough information concerning the group $\mathcal{K}$ to perform arithmetic operations in $\mathcal{K}$. (For example in threshold RSA the group is $\mathbf{Z}_{\phi(N)}$. In this case $\phi(N)$ cannot be revealed to any shareholder, otherwise they can compute the secret key d.) The shareholder can perform the arithmetic operations as integer operations. The cost of an integer addition is equal to the logarithm of the maximum addend.

Corollary 12. *The cost for a shareholder to perform the required additions to compute* $\frac{\alpha_{ax}}{\alpha_a}\boldsymbol{s}$ *is* $O(n \cdot \log_2(\max_{1 \le i \le q-2} s_i))$. *If* $\boldsymbol{s}$ *is the original share dealt to the shareholder and we are implementing the scheme to perform threshold RSA then this cost is* $O(n \log_2 \phi(N))$.

Observe that if $\boldsymbol{s}$ was the shareholder's original share then the subshare of $\frac{\alpha_{ax}}{\alpha_a}\boldsymbol{s}$ is bounded by $-q\phi(N)$ to $q\phi(N)$ (as stated by equation (9)). Hence a bound on the size of such shares would be $\log_2(q\phi(N)) = \log_2 q + \log_2 \phi(N)$. If one successively applies two distinct cyclotomic polynomials of the form $\frac{\alpha_{ax}}{\alpha_a}$ to an original share $\boldsymbol{s}$ then the bound on resulting shares would be $-q^2\phi(N)$ to $q^2\phi(N)$. Thus if one applies a product of $t-1$ cyclotomic polynomial to an original share, the bound is $-q^{t-1}\phi(N)$ to $q^{t-1}\phi(N)$.

We are now ready to describe an alternate way a shareholder may compute their partial result (signature). We will refer to this as Method $\mathcal{M}$. Method $\mathcal{M}$ requires less computational time in comparison to the method developed in [15], which we will refer to a Method $\mathcal{A}$.

Theorem 13. *Shareholder* P_i *needs to perform* $O(nt)$ *integer additions to compute* $y_{i,B}\boldsymbol{s}_i$. *In addition, the shareholder will need to perform* $O(tn \log_2 n)$ *elementary operations.*

Proof. In [15], it was established that $y_{i,B} = u^{-(t-1)i} \prod_{\substack{h \in B \\ h \neq i}} \frac{\alpha_h}{\alpha_{h-i}}$. Then $y_{i,B}\boldsymbol{s} = u^{-(t-1)i} \prod_{\substack{h \in B \\ h \neq i}} \frac{\alpha_h}{\alpha_{h-i}}\boldsymbol{s}$. For each $h \in B$, $h \neq i$, $\frac{\alpha_h}{\alpha_{h-i}} = \frac{\alpha_{x(h-i)}}{\alpha_{h-i}}$, where $x = r_{h-i}h \bmod q$. Consider

$$\prod_{\substack{h \in B \\ h \neq i}} \frac{\alpha_h}{\alpha_{h-i}}\boldsymbol{s} = \frac{\alpha_{**}}{\alpha_*}\left(\cdots\left(\frac{\alpha_{**}}{\alpha_*}\left(\frac{\alpha_{**}}{\alpha_*}\boldsymbol{s}\right)\right)\cdots\right). \tag{10}$$

This represents $t-1$ successive applications of a cyclotomic polynomial to a share. To give a definitive cost we will assume we are performing threshold RSA then the cost is

$$\begin{aligned} \text{cost} &= q\log_2\phi(N) + q(\log_2(q\phi(N)) + \cdots + q(\log_2(q^{t-2}\phi(N)) \\ &= O(q(t^2\log_2 q + t\log_2\phi(N)) \\ &= O(nt(t\log_2 n + \log_2\phi(N)). \end{aligned}$$

We then need to apply $u^{-(t-1)i}$ to equation (10), this cost is n integer additions which costs $n \cdot (\log_2(q^{t-1}\phi(N))$ which is $O(n(t\log_2 n + \log_2\phi(N))$. Overall this

cost is $O(nt(\log_2 n + \log_2 \phi(N))$. The cost of integer operations can be characterized as follows: $(t-1)$ inverses in $\mathbf{Z}_q^*$ need to be computed, i.e. r_{h-i}, $(t-1)$ multiplications need to be computed, t incrementations in $\mathbf{Z}_q^*$ by a ia for $a = h-i$. This cost can be measured as $O(t(\log_2 q)^2 + t(\log_2 q)^2 + tq \log_2 q) = O(nt \log_2 n)$.

There is one more step that the shareholder must perform and that is to exponentiate the $zero^{th}$ term, i.e. $m^{F_0(y_{i,B}\boldsymbol{s}_i)}$ (we are assuming a signature scheme). Since the size of the coefficient is bounded by $-q^{t-1}\phi(N)$ to $q^{t-1}\phi(N)$, this cost is $O((t \log n + log\phi(N))(\log N)^2)$.

Shareholder's computations

	Method $\mathcal{A}$ [15]	Method $\mathcal{M}$
Time to compute a partial share	$nt(t \log n + \log n \log \phi(N))$	$nt(t \log_2 n + \log_2 \phi(N))$
Time to exponentiate	$(t \log n + \log \phi(N))(\log N)^2$	$(t \log n + \log \phi(N))(\log N)^2$

Method $\mathcal{M}$ reflects the cost of nt additions, similar to the approach discussed in Theorem 1. However our method is $1/n$ of the amount of time required by Theorem 1. Method $\mathcal{M}$ is always superior to Method $\mathcal{A}$. However it maybe that the most time consuming operation is the exponentiation which in both methods costs the same amount of time. Note that the $nt^2 \log_2 n$ of Method $\mathcal{A}$ (in "compute a partial share") represents the cost of computing $y_{i,B}$. The $nt^2 \log_2 n$ of Method $\mathcal{M}$ represents the cost of incrementing and other $\mathbf{Z}_q$ operations. The $nt \log n \log \phi(N)$ of Method $\mathcal{A}$ (in "compute a partial share") represents the cost of the group operations and $nt \log \phi(N)$ of Method $\mathcal{M}$ represents the costs of group operations. Thus we see that the improvement in time (when comparing our method to Method $\mathcal{A}$) is a factor of $\log_2 n$.

4 How a Distributor Computes the Shares

A distributor knows enough information concerning the group $\mathcal{K}$, to perform additions in the group $\mathcal{K}$.

Theorem 14. *The cost for a distributor to perform the required additions to compute* $\frac{\alpha_{ax}}{\alpha_a}\boldsymbol{s}$ *is* $O(n \cdot \log_2(|\mathcal{K}|)$ *and* $O(n \log_2 n)$ *elementary operations. If* $\boldsymbol{s}$ *is a share dealt to the shareholder and we are implementing the scheme to perform threshold RSA then the cost for the distributor to compute* $\frac{\alpha_{ax}}{\alpha_a}\boldsymbol{s}$ *is* $O(n \log_2 \phi(N))$.

The following is a simple result which we use later.

Corollary 15. *If the distributor applies two cyclotomic polynomials to a share dealt to a shareholder then the cost of performing the additions is* $O(n \log_2(\phi(N))$.

Theorem 16. *For a distributor to compute $y_{i,B}\boldsymbol{s}$. The distributor needs to perform $O(nt)$ additions in the group $\mathcal{K}$. In terms of threshold RSA, the cost is $O(nt \log_2 \phi(N))$. In addition, the distributor will need to perform $O(tn \log_2 n)$ elementary operations.*

We are ready to describe how the distributor can compute all the shares $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$. There are two alternatives. As we will establish, the second alternative is the preferred method.

4.1 Distributor–Method 1

Recall that the shares $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ can be determined by randomly selecting vectors $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{t-1}$ from $\mathcal{K}^{q-1}$ and apply the representation given by equation (3). Then for all i, $\boldsymbol{s}_i = k + \alpha_i \boldsymbol{c}_1 + \cdots + \alpha_i^{t-1} \boldsymbol{c}_{t-1}$. Observe that we can apply Horner's algorithm, and we get

$$\boldsymbol{s}_i = \alpha_i \left(\cdots \alpha_i \left(\alpha_i \boldsymbol{c}_{t-1} + \boldsymbol{c}_{t-2} \right) \cdots + \boldsymbol{c}_1 \right) + \boldsymbol{k} \tag{11}$$

Theorem 17. *To compute $\boldsymbol{s}_i$ using equation (11) requires t multiplications of an α_i to a vector and t additions of vectors. The cost of Horner's algorithm to compute the shares for threshold RSA is $O(tn^2 \log_2 \phi(N))$.*

Proof. Using Theorem 7 the cost of computing $\boldsymbol{s}_i$ is nt additions in $\mathbf{Z}_{\phi(N)}$, which is $O(nt \log_2 \phi(N))$. The distributor need to compute all n shares. Hence a total cost of $O(tn^2 \log_2 \phi(N))$.

4.2 Distributor–Method 2

Recall the following manner in which a distributor may distribute shares. The distributor selects $t-1$ random shares $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{t-1}$. The remaining $n-t$ shares are computed using equation (1), therefore

$$\boldsymbol{s}_i = y_{i,C_i}^{-1} \boldsymbol{k} - \sum_{\substack{j \in C_i \\ j \neq i}} \frac{y_{j,C_i}}{y_{i,C_i}} \boldsymbol{s}_j \tag{12}$$

where $t \leq i \leq n$ and $C_i = \{1, 2, \ldots, t-1, i\}$.

Observe that to compute $\boldsymbol{s}_i$ (for $i = t, \ldots, n$) we need to compute $\frac{y_{j,C_i}}{y_{i,C_i}}$, for $j = 1, \ldots, t-1$, and y_{i,C_i}^{-1}. Equation (12) illustrates how a combination of the secret "$y_{i,C_i}^{-1} \boldsymbol{k}$" is salted with randomness "$\sum_{\substack{j \in C_i \\ j \neq i}} \frac{y_{j,C_i}}{y_{i,C_i}} \boldsymbol{s}_j$" (recall that in this distribution method, $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{t-1}$ are random vectors) to form the share $\boldsymbol{s}_i$.

Lemma 18. *For each i, j and B, $\frac{y_{j,B}}{y_{i,B}} = -y_{j-i,B'(j,i)}$, where $B'(j,i) = \{-i\} \cup \{h - i : h \in B,\, h \neq i\}$.*

Observe that to compute $\frac{y_{j,B}}{y_{i,B}} \boldsymbol{s}$ reduces to computing a $y_{*,**}\boldsymbol{s}$. Note that this ratio will need to be determined by the distributor $(t-1)(n-t)$ times, where $B = C_i$.

Lemma 19. *For all i and all $i \in B$, $y_{i,B}^{-1} = u^{(t-1)i} =, y_{-i,B''(i)}$, where $B''(i) = \{-i\} \cup \{h - i : h \in B,\ h \neq i\}$.*

Observe that to compute $y_{i,B}^{-1}\boldsymbol{s}$ reduces to computing a $y_{*,**}\boldsymbol{s}$.

Lemma 20. *For all $i \in \{1, \ldots, t-1\}$ and $t \leq j \leq n$*

$$\frac{y_{j+1,C_i}}{y_{i,C_i}} = -u^{j-(t-1)} \frac{\alpha_{t-1-j}\alpha_{i-j}}{\alpha_{j+1}\alpha_{i-1-j}} \frac{y_{j,C_i}}{y_{i,C_i}}$$

The table below, together with Lemma 20, illustrate how a distributor can compute successive ratios of $\frac{y_{*,*}}{y_*}$'s by starting in upper left hand corner and moving to the right, using the previous entry together with the product of two cyclotomic polynomials and a u to a power.

	1	$\cdots$	$\cdots$	$t-1$
t	$\frac{y_{1,C_t}}{y_{t,C_t}}$	$\frac{y_{2,C_{t+1}}}{y_{i,C_i}}$	$\cdots$	$\frac{y_{t-1,C_i}}{y_{i,C_i}}$
$t+1$	$\frac{y_{1,C_{t+1}}}{y_{t+1,C_{t+1}}}$	$\frac{y_{2,C_{t+1}}}{y_{t+1,C_{t+1}}}$	$\cdots$	$\frac{y_{t-1,C_{t+1}}}{y_{t+1,C_{t+1}}}$
$\vdots$	$\vdots$		$\cdots$	$\vdots$
n	$\frac{y_{1,C_n}}{y_{n,C_n}}$	$\frac{y_{2,C_n}}{y_{n,C_n}}$	$\cdots$	$\frac{y_{t-1,C_n}}{y_{n,C_n}}$

(13)

From Lemma 20 we see that from the product of two cyclotomic polynomials and a u to a power applied to $\frac{y_{1,C_i}}{y_{i,C_i}}$ generates $\frac{y_{2,C_i}}{y_{i,C_i}}$. Similarly, the product of two cyclotomic polynomials and a u to a power applied to $\frac{y_{2,C_i}}{y_{i,C_i}}$ generates $\frac{y_{3,C_i}}{y_{i,C_i}}$ and so on. For each i, j with $1 \leq j \leq t-1$ and $t \leq i \leq n$, define $\gamma_{ij} = u^{j-t-1}\frac{\alpha_{t-1-j}}{\alpha_{j+1}}\frac{\alpha_{i-j}}{\alpha_{i-1-j}}$. Then

$$\begin{aligned}
\frac{y_{2,C_i}}{y_{i,C_i}} &= \gamma_{i2}\frac{y_{1,C_i}}{y_{i,C_i}} \\
\frac{y_{3,C_i}}{y_{i,C_i}} &= \gamma_{i3}\frac{y_{2,C_i}}{y_{i,C_i}} = \gamma_{i3}\gamma_{i2}\frac{y_{1,C_i}}{y_{i,C_i}} \\
&\vdots \\
\frac{y_{j,C_i}}{y_{i,C_i}} &= \gamma_{ij}\frac{y_{j-1,C_i}}{y_{i,C_i}} = \cdots = \gamma_{ij}\cdots\gamma_{i2}\frac{y_{1,C_i}}{y_{i,C_i}}
\end{aligned}$$

For the distributor to compute $\boldsymbol{s}_i$ $(t \leq i \leq n)$, they must compute two parts, one they must compute $y_{i,B}^{-1}\boldsymbol{k}$. (By Lemma 19, this has cost $O(nt \log \phi(N) + nt \log_2 n)$.) Also, the distributor must compute $\sum_{j=1}^{t-1} \frac{y_{j,C_i}}{y_{i,C_i}}\boldsymbol{s}_j$. The distributor can compute

$$\sum_{j=1}^{t-1} \frac{y_{j,C_i}}{y_{i,C_i}}\boldsymbol{s}_j = \frac{y_{1,B}}{y_{i,B}} \left(\gamma_{i2}\left(\cdots\gamma_{it-2}\left(\gamma_{it-1}\boldsymbol{s}_{t-1} + \boldsymbol{s}_{t-2}\right)\cdots + \boldsymbol{s}_2\right) + \boldsymbol{s}_1\right). \tag{14}$$

Now note that the time to compute γ_{ij} times a share is equivalent to applying two cyclotomic polynomials on a vector then apply u to a power times a share. By Theorem 6,Theorem 14, and Corollary 15, the cost is $2n \log \phi(N) + 2n \log n = O(n(\log n + \log \phi(N))$. There are $t-1$ of these multiplications by a γ_{**} to compute. In addition, there are $t-1$ vector additions to be performed. The total running time is $O(tn(\log n + \phi(N)) + tn \log \phi(N)) = O(tn(\log n + \phi(N))$. Then we must perform a y_{1,C_i} to this resulting vector. This cost is $O(tn(\log n + \phi(N))$. Thus to compute this portion of $\boldsymbol{s}_i$ has cost $O(tn(\log n + \phi(N))$. To compute $y_{i,C_i}^{-1}\boldsymbol{k}$ is $O(nt \log_2 \phi(N) + nt \log_2 n)$. To compute both portions has cost $O(tn(\log n + \phi(N))$.

Now the distributor has to compute $\boldsymbol{s}_t, \ldots, \boldsymbol{s}_n$. Altogether, to compute all $n-t$ shares has a total cost of $O(tn(n-t)(\log n + \phi(N))$.

Distributor's computations

	Method $\mathcal{A}$ [15]	Method 1 (Horner)	Method 2
Randomness required	$(t-1)(q-1)$	$(t-1)(q-1)$	$(t-1)(q-1)$
Time to compute remaining shares	$nt^2(n-t)\times$ $\log_2 n \log_2 \phi(N)$	$tn^2\times$ $\log_2 \phi(N)$	$tn(n-t)\times$ $(\log_2 \phi(N) + \log_2 n)$

When $n \log \phi(N) < (n-t)(\log \phi(N) + \log n)$, Method 2 performs better than Method 1. Although we omit the lengthy argument, the addition of $\log_2 n$ in Method 2 can be dropped, if one wants to store previously computed ia's. This is achieved with a large memory cost.

5 Comparison with Other Schemes

Comparing Method $\mathcal{M}$ with Method $\mathcal{A}$,we have seen that in terms of Shareholder computations, Method $\mathcal{M}$ always performs better or the same than Method $\mathcal{A}$. Regarding Distributor computations Method $\mathcal{M}$ always performs better than Method $\mathcal{A}$.

Similar to what was done in [15], we compare the performance of Shoup signature scheme [20] with the Desmedt-Frankel scheme using our method. Regarding Shareholder computations, there are various cases to consider. It is clear that Method $\mathcal{M}$ is superior to Shoup's scheme (regarding Shareholder computations) when $(t \log_2 n + \log_2 \phi(N))(\log_2 N)^2 \geq nt(t \log_2 n + \log_2 \phi(N))$ Regarding Distributor computations, Method $\mathcal{M}$ is superior to Shoup's scheme whenever $(\log_2 \phi(N))^2 \geq (n-t)(\log_2 n + \log_2 \phi(N))$. Thus in many cases the Desmedt-Frankel scheme performs better than other threshold RSA signature scheme. This scheme still has disadvantages, in that it has large shares and it requires a large amount of randomness for the distributor to compute the shares. Shoup's scheme, like the schemes by Rabin [17] and Frankel, Gemmel, Mackenzie and Yung [12], may perform worse than the Desmedt-Frankel scheme, because they utilize a technique of exponentiating to a large exponent, an exponent greater

than $n!$, rather than using expanded shares. Whereas in the Desmedt-Frankel scheme the largest the exponent can be is $q^t\phi(N)$. In reality $q^t\phi(N)$ is an upper bound and is dependent on the i and B. In many cases, the exponent will be much smaller. We do note that threshold schemes like [12] have other benefits that the Desmedt-Frankel scheme does not possess.

	Shoup's scheme	Desmedt-Frankel scheme Method $\mathcal{M}$
Size of share	1	n
Shareholder time required	$(n\log_2 n + \log_2 \phi(N))\times (\log_2 N)^2$	$max\{nt(t\log_2 n + \log_2 \phi(N)), [t\log_2 n + \log_2 \phi(N)](\log_2 N)^2\}$
Distributor randomness required	$t-1$	$(t-1)O(n)$
Distributor time required	$nt(\log_2 \phi(N))^2$	$nt(n-t)(\log_2 n + \log_2 \phi(N))$
Combiner time required	$(t + \log_2 a + \log_2 b)\times (\log_2 N)^2$	$t(\log_2 N)^2$

6 Conclusion

We have described algorithms which effectively speed up computations in the threshold RSA scheme developed by [4]. We have pointed out that there are many occasions when the Desmedt-Frankel scheme will perform better than "efficient" schemes. However, our work has not reduced share size nor the randomness required. It is ironic that with a large share, computations may be maybe smaller than schemes with small share size. In the end, in most threshold examples, it is response time, i.e. time required for the shareholder to respond to a signature request, that is most important. Further, we point out that although shares are large, the partial signatures sent to the Combiner are the same size as an RSA signature. If the distributor has enough random resources, it appears that the remaining computations is comparable (or better) than distributor work for other threshold schemes. Lastly the Combiner's amount of computations within the Desmedt-Frankel scheme, are less than or equal, to the Combiner's work in Shoup's scheme.

References

1. W. Adkins and S. Weintrab. *Algebra, an approach via module theory.* Springer-Verlag, NY, 1992.
2. G. Blakley. "Safeguarding cryptographic keys." In *Proc. Nat. Computer Conf. AFPIPS Conf. Proc., 48* pp. 313-317, 1979.
3. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. "How to share a function". In *Proceedings of the twenty-sixth annual ACM Symp. Theory of Computing (STOC)*, pp. 522-533, 1994.

4. Y. Desmedt and Y. Frankel. "Homomorphic zero-knowledge threshold schemes over any finite abelian group". In *Siam J. Disc. Math. vol 7, no. 4* pp. 667-679, SIAM, 1994.
5. Y. Desmedt and Y. Frankel. Threshold Cryptosystems *In Advances of Cryptology- Crypto '89*, pp. 307-315, 1989.
6. Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In *Advances of Cryptology- Crypto '91*, 1991.
7. Y. Desmedt and S. Jajodia. Redistributing secret shares to new access structures and its applications. Tech. Report ISSE-TR-97-01, George Mason University, July 1997 ftp://isse.gmu.edu/pub/techrep/97.01.jajodia.ps.gz
8. Y. Desmedt, B. King, W. Kishimoto, and K. Kurosawa, "A comment on the efficiency of secret sharing scheme over any finite abelian group", In *Information Security and Privacy*, ACISP'98 (Third Australasian Conference on Information Security and Privacy), LNCS 1438, 1998, 391-402.
9. Y.Frankel. A practical protocol for large group oriented networks. In *Advances of Cryptology- Eurocrypt '89*, Lecture Notes in Computer Science 434, Springer Verlag, 1990, pp 56-61.
10. Y.Frankel and Y. Desmedt. Parallel reliable threshold multisignature. *Tech. report TR-92-04-02*, Univ. of Wisconsin-Milwaukee, 1992.
11. Y. Frankel, P. Gemmel, P. Mackenzie, and M. Yung. Proactive RSA In *Advances of Cryptology- Crypto '97*, 1997 Lecture Notes in Computer Science 1294, Springer Verlag, 1997, pp 440-454.
12. Y. Frankel, P. Gemmel, P. Mackenzie, and M. Yung. Optimal-Resilience Proactive Public-key Cryptosystems In *Proc. 38th FOCS*, pp 384-393, IEEE, 1997
13. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In *Advances of Cryptology- Crypto '96,* Lecture Notes in Computer Science 1109, Springer Verlag, 1996, pp 157-172.
14. T. Hungerford. *Algebra.* Springer-Verlag, NY, 1974.
15. B. King. Algorithms to speed up computations in threshold RSA, Australasian Conference on Information Security and Privacy 2000.
16. H.L. Keng. *Introduction to Number Theory.* Springer Verlag, NY 1982
17. T.Rabin. A Simplified Approach to threshold and proactive RSA. In *Advances of Cryptology- Crypto '98,* 1998
18. R. Rivest, A. Shamir, and L. Adelman, A method for obtaining digital signatures and public key cryptosystems, *Comm. ACM,* 21(1978), pp 294-299.
19. A. Shamir, How to share a secret, *Comm. ACM*, 22(1979), pp 612-613.
20. V. Shoup. "Practical Threshold Signatures" In *Advances of Cryptology- Eurocrypt 2000,* pp 207-220.