# Threshold Cryptography Based on Asmuth-Bloom Secret Sharing

Kamer Kaya, Ali Aydın Selçuk

*Department of Computer Engineering*
*Bilkent University*
*Ankara, 06800, Turkey*
{*kamer,selcuk*}*@cs.bilkent.edu.tr*

**Abstract**

In this paper, we investigate how threshold cryptography can be conducted with the Asmuth-Bloom secret sharing scheme and present three novel function sharing schemes for RSA, ElGamal and Paillier cryptosystems. To the best of our knowledge, these are the first provably secure threshold cryptosystems realized using the Asmuth-Bloom secret sharing. The proposed schemes are efficient in terms of performance and compare favorably to earlier proposals.

*Key words:* Threshold cryptography, function sharing schemes, Asmuth-Bloom secret sharing, RSA, ElGamal, Paillier

## 1 Introduction

Threshold cryptography deals with the problem of sharing a highly sensitive secret among a group of $n$ users so that only when a sufficient number $t$ of them come together can the secret be reconstructed. Well-known secret sharing schemes (SSS) in the literature include Shamir [18] based on polynomial interpolation, Blakley [6] based on hyperplane geometry, and Asmuth-Bloom [3] based on the Chinese Remainder Theorem.

[1] This work is supported in part by the Turkish Scientific and Technological Research Agency (TÜBİTAK), under grant number EEEAG-105E065.
[2] A preliminary version of this paper was presented in ISCIS'06, 21st International Symposium on Computer and Information Sciences

A further requirement of a threshold cryptosystem can be that the subject function (e.g., a digitial signature) should be computable without the involved parties disclosing their secret shares. This is known as the *function sharing problem*. A function sharing scheme (FSS) requires distributing the function's computation according to the underlying SSS such that each part of the computation can be carried out by a different user and then the partial results can be combined to yield the function's value without disclosing the individual secrets. Several protocols for function sharing [7–9,11,19] have been proposed in the literature. Nearly all existing solutions for function sharing have been based on the Shamir SSS [18].

In this paper, we show how sharing of cryptographic functions can be securely achieved using the Asmuth-Bloom secret sharing scheme. We give three novel FSSs, one for the RSA [17], one for the ElGamal decryption [12] and the other for the Paillier decryption [16] functions. These public key cryptosystems have various properties exploited by several applications [1,4,13–15]. The proposed schemes are provably secure and to the best of our knowledge they are the first realization of function sharing based on the Asmuth-Bloom SSS.

The organization of the paper is as follows: In Section 2, we give an overview of threshold cryptography and review the existing secret and function sharing schemes in the literature. We discuss the Asmuth-Bloom SSS in detail in Section 3 and the modifications on the basic scheme in Section 4. In Sections 5, 6, 7, we describe the FSSs for RSA, ElGamal and Paillier cryptosystems, respectively, and prove their security features. We conclude the paper with an assessment of the proposed schemes in Section 8.

## 2 Background

Constructing threshold schemes for secret and function sharing is the main research area in threshold cryptography. These problems were extensively analyzed and several solutions were proposed in the literature.

### *2.1 Secret Sharing Schemes*

The problem of secret sharing and the first solutions to it were introduced independently by Shamir [18] and Blakley [6] in 1979. A $(t, n)$-secret sharing scheme is used to distribute a secret $d$ among $n$ people such that any coalition of size $t$ or more can construct $d$ but smaller coalitions cannot. Furthermore, an SSS is said to be *perfect* if coalitions smaller than $t$ cannot obtain *any* information on $d$; i.e., the candidate space for $d$ cannot be reduced even by

one candidate by using $t-1$ or fewer shares.

The first scheme for sharing a secret was proposed by Shamir [18] based on polynomial interpolation. To obtain a $(t, n)$ secret sharing, a random polynomial $f(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \ldots + a_0$ is generated over $\mathbb{Z}_p[x]$ where $p$ is a prime number and $a_0 = d$ is the secret. The share of the $i$th party is $y_i = f(i)$, $1 \leq i \leq n$. If $t$ or more parties come together, they can construct the polynomial by Lagrangian interpolation and obtain the secret, but any smaller coalitions cannot.

Another interesting SSS is the scheme proposed by Blakley [6]. In a $t$ dimensional space, a system of $t$ non-parallel, non-degenerate hyperplanes intersect at a single point. In Blakley's scheme, a point in the $t$ dimensional space (or, its first coordinate) is taken as the secret and each party is given a hyperplane passing through that point. When $t$ users come together, they can uniquely identify the secret point, but smaller coalitions cannot.

A fundamentally different SSS is the scheme of Asmuth and Bloom [3], which shares a secret among the parties using modular arithmetic and reconstructs it by the Chinese Remainder Theorem. We describe this scheme in detail in Section 3.

### *2.2 Function Sharing Schemes*

Function sharing schemes were first introduced by Desmedt et al. [8] in 1989. In a $(t, n)$ function sharing scheme, a key-dependent function is distributed among $n$ people such that any coalition of size $t$ or more can evaluate the function but smaller coalitions cannot. When a coalition $\mathcal{S}$ is to evaluate the function, the $i$th user in $\mathcal{S}$ computes his own partial result by using his share $y_i$ and sends it to a platform which combines these partial results. Unlike in a secret sharing scheme, the platform here need not be trusted since the user shares are not disclosed to the platform.

FSSs are typically used to distribute the private key operations in a public key cryptosystem (i.e., the decryption and signature operations) among several parties. Sharing a private key operation in a threshold fashion requires first choosing a suitable SSS to share the private key. Then the subject function must be arranged according to this SSS such that combining the partial results from any $t$ parties will yield the operation's result correctly. This is usually a challanging task and requires some ingenious techniques.

Several solutions for sharing the RSA, ElGamal and Paillier private key operations have been proposed in the literature [2,7–11,13,19]. Almost all of these schemes are based on the Shamir SSS, with the only exception of one scheme

in [8] based on Blakley. Lagrangian interpolation used in the secret reconstruction phase of Shamir's scheme makes it a suitable choice for function sharing, but it also provides several challenges. One of the most significant challenges is the computation of inverses in $\mathbb{Z}_{\phi(N)}$ for sharing the RSA function where $\phi(N)$ should not be known by the users. The first solution to this problem was proposed by Desmedt and Frankel [7], which solved the problem by making the dealer compute all potentially needed inverses at the setup time and distribute them to users mixed with the shares. A more elegant solution was found a few years later by De Santis et al. [11]. They carried the arithmetic into a cyclotomic extension of $\mathbb{Z}$, which enabled computing the inverses without knowing $\phi(N)$. Finally, a very practical and ingenious solution was given by Shoup [19] where he removed the need of taking inverses in Lagrangian interpolation altogether.

To the best of our knowledge, so far no secure function sharing schemes based on the Asmuth-Bloom SSS have been proposed in the literature. We show in this paper that the Asmuth-Bloom scheme in fact can be a more suitable choice for function sharing than its alternatives, and the fundamental challanges of function sharing with other SSSs do not exist for the Asmuth-Bloom scheme.

## 3 Asmuth-Bloom Secret Sharing Scheme

In the Asmuth-Bloom SSS, sharing and reconstruction of the secret are done as follows:

- *Sharing Phase:* To share a secret $d$ among a group of $n$ users, the dealer does the following:

(1) A set of pairwise relatively prime integers $m_0 < m_1 < m_2 < \ldots < m_n$, where $m_0 > d$ is a prime, are chosen such that

$$\prod_{i=1}^{t} m_i > m_0 \prod_{i=1}^{t-1} m_{n-i+1}. \tag{1}$$

(2) Let $M$ denote $\prod_{i=1}^{t} m_i$. The dealer computes

$$y = d + Am_0$$

where $A$ is a positive integer generated randomly subject to the condition that $0 \leq y < M$.

(3) The share of the $i$th user, $1 \leq i \leq n$, is

$$y_i = y \bmod m_i.$$

- *Construction Phase:*

Assume $\mathcal{S}$ is a coalition of $t$ users to construct the secret. Let $M_{\mathcal{S}}$ denote $\prod_{i\in\mathcal{S}} m_i$.

(1) Given the system

$$y \equiv y_i \pmod{m_i}$$

for $i \in \mathcal{S}$, solve $y$ in $\mathbb{Z}_{M_{\mathcal{S}}}$ using the Chinese Remainder Theorem.

(2) Compute the secret as

$$d = y \bmod m_0.$$

According to the Chinese Remainder Theorem, $y$ can be determined uniquely in $\mathbb{Z}_{M_{\mathcal{S}}}$. Since $y < M \leq M_{\mathcal{S}}$ the solution is also unique in $\mathbb{Z}_M$.

The Asmuth-Bloom SSS is a perfect sharing scheme: Assume a coalition $\mathcal{S}'$ of size $t-1$ has gathered and let $y'$ be the unique solution for $y$ in $Z_{M_{\mathcal{S}'}}$. According to (1), $M/M_{\mathcal{S}'} > m_0$, hence $y' + jM_{\mathcal{S}'}$ is smaller than $M$ for $j < m_0$. Since $\gcd(m_0, M_{\mathcal{S}'}) = 1$, all $(y' + jM_{\mathcal{S}'}) \bmod m_0$ are distinct for $0 \leq j < m_0$, and there are $m_0$ of them. That is, $d$ can be any integer from $\mathbb{Z}_{m_0}$, and the coalition $\mathcal{S}'$ obtains no information on $d$.

## 4 Function Sharing Based on the Asmuth-Bloom Scheme

There are several changes we have made on the basic Asmuth-Bloom scheme to make it more suitable for function sharing schemes. In this section we describe these modifications.

In the original Asmuth-Bloom SSS, the authors proposed a recursive process to solve the system $y \equiv y_i \pmod{m_i}$. Instead, we use a direct solution which is more suitable for function sharing. Suppose $\mathcal{S}$ is a coalition of $t$ users gathered to construct the secret $d$.

(1) Let $M_{\mathcal{S}\setminus\{i\}}$ denote $\prod_{j\in S, j\neq i} m_j$ and $M'_{\mathcal{S},i}$ be the multiplicative inverse of $M_{\mathcal{S}\setminus\{i\}}$ in $\mathbb{Z}_{m_i}$, i.e.,

$$M_{\mathcal{S}\setminus\{i\}} M'_{\mathcal{S},i} \equiv 1 \pmod{m_i}.$$

First, the $i$th user computes

$$u_i = y_i M'_{\mathcal{S},i} M_{\mathcal{S}\setminus\{i\}} \bmod M_{\mathcal{S}}.$$

(2) $y$ is computed as

$$y = \sum_{i\in\mathcal{S}} u_i \bmod M_{\mathcal{S}}. \tag{2}$$

(3) The secret $d$ is computed as

$$d = y \bmod m_0.$$

In the Asmuth-Bloom SSS, $m_0$ need not be a prime, and the scheme works correctly for a composite $m_0$ as long as $m_0$ is relatively prime to $m_i$, $1 \leq i \leq n$. Also note that $m_0$ need not be known during the secret construction process until the $3^{rd}$ step above. In the FSSs described below, $m_i$, $1 \leq i \leq n$, are known by all users, but $m_0$ is kept secret by the dealer unless otherwise is stated.

We also modified (1) as

$$\prod_{i=1}^{t} m_i > {m_0}^2 \prod_{i=1}^{t-1} m_{n-i+1}. \tag{3}$$

in order to use it securely in the proposed FSSs.

## 5 Sharing of the RSA Function

RSA [17] is the first and most commonly used public key cryptosystem for signing and encrypting messages and still it is used in numerous applications requiring public key cryptography. The RSA signature and decryption functions are identical and sharing one of them enables sharing the other. Here, we discuss how to share the RSA signature function. The same techniques apply to the decryption function as well.

- *Setup*: Let $N = pq$ be the product of two large prime numbers. Choose a random $e \in \mathbb{Z}^*_{\phi(N)}$ and find its inverse $d$, i.e., $ed \equiv 1 \pmod{\phi(N)}$. $(N, e)$ and $d$ are the public and private keys, respectively.
- *Signing*: Given a hashed message $w \in \mathbb{Z}_N$, the signature $s$ is computed as

$$s = w^d \bmod N. \tag{4}$$

- *Verification*: Given a signature $s \in \mathbb{Z}_N$, the verification is done by checking

$$w \stackrel{?}{=} s^e \bmod N. \tag{5}$$

The following is a procedure describes how the RSA signature function can be shared with the Asmuth-Bloom SSS:

(1) In threshold RSA setup, choose the RSA primes $p = 2p'+1$ and $q = 2q'+1$ where $p'$ and $q'$ are also large random primes. $N = pq$ is computed and the public key $e$ and private key $d$ are chosen from $\mathbb{Z}^*_{\phi(N)}$ where $ed \equiv 1 \pmod{\phi(N)}$. Use Asmuth-Bloom SSS for sharing $d$ with $m_0 = \phi(N) = 4p'q'$.

(2) Let $w$ be the hashed message to be signed and suppose the range of the hash function is $Z^*_N$. Assume a coalition $\mathcal{S}$ of size $t$ wants to obtain the

signature $s = w^d \bmod N$. The $i$th user in the coalition knows $m_j$ for all $j \in \mathcal{S}$ and $y_i = y \bmod m_i$ as its secret share.

(3) Each user $i \in \mathcal{S}$ computes

$$u_i = y_i M'_{\mathcal{S},i} M_{\mathcal{S}\setminus\{i\}} \bmod M_{\mathcal{S}}, \tag{6}$$

$$s_i = w^{u_i} \bmod N. \tag{7}$$

(4) The *incomplete signature* $\overline{s}$ is obtained by combining the $s_i$ values

$$\overline{s} = \prod_{i \in \mathcal{S}} s_i \bmod N. \tag{8}$$

(5) Let $\kappa = w^{-M_{\mathcal{S}}} \bmod N$ be the *corrector*. The partial signature can be corrected by trying

$$(\overline{s}\kappa^j)^e = \overline{s}^e (\kappa^e)^j \stackrel{?}{\equiv} w \pmod N \tag{9}$$

for $0 \leq j < t$. Then the signature $s$ is computed by

$$s = \overline{s}\kappa^\delta \bmod N$$

where $\delta$ denotes the value of $j$ that satisfies (9).

We call the signature $\overline{s}$ generated in (8) *incomplete* since we need to obtain $y = \sum_{i \in \mathcal{S}} u_i \bmod M_{\mathcal{S}}$ as the exponent of $w$. Once this is achieved, we have $w^y \equiv w^d \pmod N$ as $y = d + Am_0$ for some $A$ and we chose $m_0 = \phi(N)$.

Note that the equality in (9) must hold for some $j \leq t-1$ since the $u_i$ values were already reduced modulo $M_{\mathcal{S}}$. So, combining $t$ of them in (8) will give $d + am_0 + \delta M_{\mathcal{S}}$ in the exponent for some $\delta \leq t-1$. Thus in (8), we obtained

$$\overline{s} = w^{d+\delta M_{\mathcal{S}}} \bmod N \equiv s w^{\delta M_{\mathcal{S}}} \bmod N \equiv s\kappa^{-\delta} \bmod N$$

and for $j = \delta$, equation (9) will hold. Also note that the mappings $w^e \bmod N$ and $w^d \bmod N$ are bijections in $\mathbb{Z}_N$, hence there will be a unique value of $s = \overline{s}\kappa^j$ which satisfies (9).

### *5.1 Security Analysis*

Here we will prove that the proposed threshold RSA scheme is secure, i.e. existentially non-forgeable against an adaptive chosen message attack, provided that the RSA problem is intractable. Throughout the paper, we assume a static adversary model where the adversary controls exactly $t-1$ users and chooses them at the beginning of the attack. In this model, the adversary obtains all secret information of the corrupted users and the public parameters of the cryptosystem. She can control the actions of the corrupted users, ask for

signatures of the messages chosen by herself but she cannot corrupt another user in the course of an attack, i.e., the adversary is static in that sense.

**Theorem 5.1** *If an adversary who controls $t-1$ users can forge signatures in the threshold RSA scheme, she can also forge signatures in the standard RSA scheme.*

**Proof** We will show that if the standard RSA scheme is secure the threshold RSA scheme is also secure. To do this, we will simulate the threshold protocol with no information on the secret where the output of the simulator is indistinguishable from the adversary's point of view. After that, we will show that, the secrecy of the private key $d$ is not disrupted by the values obtained by the adversary. Thus, if the threshold RSA scheme is not secure, i.e., an adversary who controls $t-1$ users can forge signatures in the threshold scheme, one can use this simulator to forge a signature in the standard RSA scheme.

Let $\mathcal{S}'$ denote the set of users controlled by the adversary. To simulate the adversary's view, a random $y$ value is chosen from $\mathbb{Z}_M$, $M = \prod_{i=1}^{t} m_i$. Then, the shares of the corrupted users are computed by $y_j = y \bmod m_j$ for $j \in \mathcal{S}'$. Note that, in the threshold RSA signature scheme, $y$ can be any integer in $\mathbb{Z}_M$ so the distribution of these compromised shares is indistinguishable from the distribution of the shares in the real case.

Since we have a $(t, n)$-threshold scheme, given a valid RSA signature $(s, w)$, the partial signature $s_i$ for a user $i \notin \mathcal{S}'$ can be obtained by

$$s_i = s\kappa^{-\delta_{\mathcal{S}}} \prod_{j \in \mathcal{S}'} (w^{u_j})^{-1} \bmod N \tag{10}$$

where $\mathcal{S} = \mathcal{S}' \cup \{i\}$, $\kappa = w^{-M_{\mathcal{S}}} \bmod N$ and $\delta_{\mathcal{S}}$ is equal to either $\left\lfloor \frac{\sum_{j \in \mathcal{S}'} u_j}{M_{\mathcal{S}}} \right\rfloor + 1$ or $\left\lfloor \frac{\sum_{j \in \mathcal{S}'} u_j}{M_{\mathcal{S}}} \right\rfloor$. To determine the value of $\delta_{\mathcal{S}}$, the simulator acts according to the $y$ value randomly chosen at the beginning of the simulation. The value of $\delta_{\mathcal{S}}$ is important because it carries some information on $y$. Let $U = \sum_{j \in \mathcal{S}'} u_j$ and $U_{\mathcal{S}} = U \mod M_{\mathcal{S}}$. One can find whether $y$ is greater than $U_{\mathcal{S}}$ or not by looking $\delta_{\mathcal{S}}$:

$$\delta_{\mathcal{S}} = \begin{cases} \lfloor U/M_{\mathcal{S}} \rfloor + 1, & \text{if} \quad y < U_{\mathcal{S}} \\ \lfloor U/M_{\mathcal{S}} \rfloor, & \text{if} \quad y \geq U_{\mathcal{S}} \end{cases} \tag{11}$$

The simulator also chooses the value of $\delta_{\mathcal{S}}$ according to (11).

Now, we will prove that the $\delta_{\mathcal{S}}$ values computed by the simulator does not disrupt the secrecy of the private key $d$ from the adversary's point of view. First of all, there are $(n - t + 1)$ possible $\delta_{\mathcal{S}}$ computed by using $U_{\mathcal{S}}$ since all the operations in the exponent depend only on the coalition $\mathcal{S}$. Now, consider an interval $[a, b) \subset [0, M)$ where $a \leq y < b$. It is obvious that if none of

these $U_{\mathcal{S}}$ values lies in that interval, choosing another integer $\overline{y}$ from $[a, b)$ such that $y_j = \overline{y} \bmod m_j$ for all $j \in \mathcal{S}'$ does not change the $\delta_{\mathcal{S}}$ values. Using this observation, we can prove that no information about the private key is obtained by the adversary.

Consider the interval $I = [y, y + m_0 M_{\mathcal{S}'})$. There are $m_0 = \phi(N)$ candidates for $y$ in $I$ which satisfy $y_j = y \bmod m_j$ for all $j \in \mathcal{S}'$. These $m_0$ candidates have all different remainders modulo $m_0$ since $\gcd(M_{\mathcal{S}'}, m_0) = 1$. So, exactly one of the remainders is equal to the private key $d$. If $U_{\mathcal{S}} \notin I$ for all $\mathcal{S}$, given an $s_i$, the shared value $y$ can be equal to any of these $m_0$ candidates hence the secret space for the private key remains same. Fortunately, this happens with all but negligible probability. The probability of $U_{\mathcal{S}} \notin I$ for a coalition $\mathcal{S}$ is equal to $\left(1 - \frac{m_0 M_{\mathcal{S}'}}{M_{\mathcal{S}}}\right)$. According to (3), the probability of $U_{\mathcal{S}} \notin I$ for all possible $\mathcal{S}$ is less than $\left(1 - \frac{1}{m_0}\right)^{n-t+1}$, which is almost surely equal to 1 since $m_0 \gg n$.

According to the above arguments, the output of the simulator is indistinguishable from the adversary's point of view, and hence the simulator can be used to break the standard RSA scheme if the threshold RSA scheme is not secure. □

## 6 Sharing of the ElGamal Decryption Function

ElGamal is another popular public key cryptosystem proposed in 1984 by T. ElGamal [12]. It is a semantically secure and inherently probabilistic encryption scheme which also supports homomorphic encryption. The description of the cryptosystem is as follows:

- *Setup*: Let $p$ be a large prime and $g$ be a generator of $\mathbb{Z}_p$. Choose a random $\alpha \in \{1, \ldots, p-1\}$ and compute $\beta = g^\alpha \bmod p$. $(\beta, g, p)$ and $\alpha$ are the public and private keys, respectively.
- *Encryption*: Given a message $w \in \mathbb{Z}_p$, the ciphertext $c$ is computed as

$$c = (c_1 = g^r \bmod p, c_2 = \beta^r w \bmod p) \tag{12}$$

  where $r$ is a random integer from $\mathbb{Z}_p$.
- *Decryption*: Given a ciphertext $c$, the message $w$ is computed as

$$w = ({c_1}^\alpha)^{-1} c_2 \bmod p. \tag{13}$$

Both RSA and ElGamal encryption scheme has a homomorphic property desirable for some applications:

$$E(w_1)E(w_2) = E(w_1 w_2) \tag{14}$$

for messages $w_1$ and $w_2$ where $E$ stands for the encryption function. Since the standard RSA encryption is not a probabilistic scheme, it is not semantically secure. One can use random padding to obtain a probabilistic scheme as in [5]. However, this removes the homomorphic property. ElGamal does not suffer from such a problem since it is already semantically secure. This property makes ElGamal eligible to be used in threshold authenticated key exchange protocols [1].

The following is a procedure describes how the ElGamal decryption function can be shared with the Asmuth-Bloom SSS:

(1) In threshold ElGamal setup, choose $p = 2q+1$ where $q$ is a large random prime and let $g$ be a generator of $\mathbb{Z}_p^*$. Choose a random $\alpha \in \{1, \ldots, p-1\}$ and compute $\beta = g^\alpha \bmod p$. Let $\alpha$ and $(\beta, g, p)$ be the private and the public key, respectively. Use Asmuth-Bloom SSS for sharing the private key $\alpha$ with $m_0 = 2q$.

(2) Let $(c_1, c_2)$ be the ciphertext to be decrypted where $c_1 = g^k \bmod p$ for some $k \in \{1, \ldots, p-1\}$ and $c_2 = \beta^k w$ where $w$ is the message. The coalition $\mathcal{S}$ of $t$ users wants to obtain the message $w = sc_2 \bmod p$ for the *decryptor* $s = (c_1^\alpha)^{-1} \bmod p$. The $i$th user in the coalition knows $m_j$ for all $j \in \mathcal{S}$ and $y_i = y \bmod m_i$ as its secret share.

(3) Each user $i \in \mathcal{S}$ computes

$$u_i = y_i M'_{\mathcal{S},i} M_{\mathcal{S}\setminus\{i\}} \bmod M_{\mathcal{S}}, \tag{15}$$

$$s_i = {c_1}^{-u_i} \bmod p, \tag{16}$$

$$\beta_i = g^{u_i} \bmod p. \tag{17}$$

(4) The *incomplete decryptor* $\overline{s}$ is obtained by combining the $s_i$ values

$$\overline{s} = \prod_{i \in \mathcal{S}} s_i \bmod p. \tag{18}$$

(5) The $\beta_i$ values will be used to find the exponent which will be used to correct the incomplete decryptor. Compute the incomplete public key $\overline{\beta}$ as

$$\overline{\beta} = \prod_{i \in \mathcal{S}} \beta_i \bmod p. \tag{19}$$

Let $\kappa_s = {c_1}^{M_{\mathcal{S}}} \bmod p$ and $\kappa_\beta = g^{-M_{\mathcal{S}}} \bmod p$ be the *correctors* for $s$ and $\beta$, respectively. The corrector exponent $\delta$ can be obtained by trying

$$\overline{\beta} \kappa_\beta^j \stackrel{?}{\equiv} \beta \pmod p \tag{20}$$

for $0 \leq j < t$.

(6) Compute the message $w$ as

$$s = \overline{s} {\kappa_s}^\delta \bmod p, \tag{21}$$

$$w = sc_2 \bmod p. \tag{22}$$

where $\delta$ denotes the value for $j$ that satisfies (20).

As in the case of RSA, the decryptor $\overline{s}$ is *incomplete* since we need to obtain $y = \sum_{i \in \mathcal{S}} u_i \bmod M_{\mathcal{S}}$ as the exponent of $c_1^{-1}$. Once this is achieved, $(c_1^{-1})^y \equiv (c_1^{-1})^\alpha \pmod N$ since $y = \alpha + A\phi(p)$ for some $A$.

When the equality in (20) holds we know that $\beta = g^\alpha \bmod p$ is the correct public key. This equality must hold for one $j$ value, denoted by $\delta$, in the given interval because since the $u_i$ values in (15) and (17) are first reduced modulo $M_{\mathcal{S}}$. So, combining $t$ of them will give $\alpha + am_0 + \delta M_{\mathcal{S}}$ in the exponent in (19) for some $\delta \leq t-1$. Thus in (19), we obtained

$$\overline{\beta} = g^{\alpha + am_0 + \delta M_{\mathcal{S}}} \bmod p \equiv g^{\alpha + \delta M_{\mathcal{S}}} \equiv \beta g^{\delta M_{\mathcal{S}}} \equiv \beta \kappa_\beta^{-\delta} \pmod p$$

and for $j = \delta$ equality must hold. Actually, in (19) and (20), our purpose is not computing the public key since it is already known. We want to find the corrector exponent $\delta$ to obtain $s$, which is also equal to one we use to obtain $\beta$. The equality can be verified as seen below:

$$\begin{aligned} s \equiv {c_1}^{-\alpha} &\equiv \beta^{-r} && \pmod p \\ &\equiv \left(g^{-(\alpha + (\delta - \delta) M_{\mathcal{S}})}\right)^r && \pmod p \\ &\equiv {c_1}^{-(\alpha + am_0 + \delta M_{\mathcal{S}})} \left({c_1}^{M_{\mathcal{S}}}\right)^\delta \equiv \overline{s} {\kappa_s}^\delta && \pmod p \end{aligned}$$

### *6.1 Security Analysis*

Here, we will prove that the threshold ElGamal encryption scheme is semantically secure against under the static adversary model provided that the ElGamal encryption scheme is also semantically secure. The threshold semantic security definition and the attack model are taken from [13].

**Theorem 6.1** *If the threshold ElGamal encryption scheme is not semantically secure against a static adversary who controls exactly $t-1$ users, the standard ElGamal scheme is not semantically secure.*

**Proof** The structure of the proof is similar to the one we did for the threshold RSA signature scheme.

Let $\mathcal{S}'$ denote the set of users controlled by the adversary. To simulate the adversary's view, a random $y$ value is chosen from $\mathbb{Z}_M$, $M = \prod_{i=1}^t m_i$. Then, the compromised shares are computed by $y_j = y \bmod m_j$ for $j \in \mathcal{S}'$. Note that, in the threshold ElGamal signature scheme, $y$ can be any integer in $\mathbb{Z}_M$ so the distribution of the shares of corrupted users is indistinguishable from the distribution of the shares in the real case.

Since we have a $(t, n)$-threshold scheme, when we determine the $y_j$ values for $j \in \mathcal{S}'$, the shares of other users are also determined. Although they cannot be computed easily, given a valid message-ciphertext pair $(w, (c_1, c_2))$ the partial decryptor $s_i$ and $\beta_i$ for a user $i \notin \mathcal{S}'$ can be obtained by

$$s_i = \left(w{c_2}^{-1}\right) {\kappa_s}^{-\delta_{\mathcal{S}}} \prod_{j \in \mathcal{S}'} {c_1}^{u_j} \bmod p, \tag{23}$$

$$\beta_i = \beta {\kappa_\beta}^{-\delta_{\mathcal{S}}} \prod_{j \in \mathcal{S}'} (\beta^{u_j})^{-1} \bmod p. \tag{24}$$

where $\mathcal{S} = \mathcal{S}' \cup \{i\}$, $\kappa_s = {c_1}^{M_{\mathcal{S}}} \bmod p$, $\kappa_\beta = g^{-M_{\mathcal{S}}} \bmod p$ and $\delta_{\mathcal{S}}$ is equal to either $\lfloor \frac{\sum_{j \in \mathcal{S}'} u_j}{M_{\mathcal{S}}} \rfloor + 1$ or $\lfloor \frac{\sum_{j \in \mathcal{S}'} u_j}{M_{\mathcal{S}}} \rfloor$. We use the same ideas to choose the value of $\delta_{\mathcal{S}}$ as in the previous simulator so we skip the details and the analysis for the secrecy of the private key in the proof.

Consequently, the output of the simulator is indistinguishable from the adversary's point of view, and hence we proved that the threshold ElGamal scheme must be semantically secure if the standard one is. □

## 7 Sharing of the Paillier Decryption Function

Different from the previously discussed public key cryptosystems, Paillier's probabilistic cryptosystem is a member of another class of cryptosystems, that use the message in the exponent of the encryption operation [16]. The description of the cryptosystem is as follows:

- *Setup*: Let $N = pq$ be the product of two large prime numbers and $\lambda = lcm(p-1, q-1)$. Choose a random $g \in \mathbb{Z}_{N^2}$ such that the order of $g$ is a multiple of $N$. $(N, g)$ and $\lambda$ are the public and private keys, respectively.
- *Encryption*: Given a message $w \in \mathbb{Z}_N$, the ciphertext $c$ is computed as

$$c = g^w r^N \bmod N^2 \tag{25}$$

  where $r$ is a random number from $\mathbb{Z}_N$.
- *Decryption*: Given a ciphertext $c \in \mathbb{Z}_{N^2}$, the message $w$ is computed as

$$w = \frac{L\left(c^\lambda \bmod N^2\right)}{L\left(g^\lambda \bmod N^2\right)} \bmod N \tag{26}$$

  where $L(x) = \frac{x-1}{N}$.

Paillier's encryption scheme is probabilistic and has interesting homomorphic properties:

$$E(w_1)E(w_2) = E(w_1 + w_2) \tag{27}$$
$$E(w)^a = E(aw) \tag{28}$$

for messages, $w, w_1, w_2$ and integer $a$. These homomorphic properties make this encryption scheme suitable for different applications such as secure voting and lottery protocols [4,13], DSA sharing protocols [14], and private information retrieval [15].

The following is a procedure describes how the Pailier decryption function can be shared with the Asmuth-Bloom SSS. The setup part (1) is inspired by [13]:

(1) In threshold Paillier setup, choose large primes $p = 2p'+1$ and $q = 2q'+1$ where $p'$ and $q'$ are also large random primes and $\gcd(N, \phi(N)) = 1$ for $N = pq$. Let $g = (1 + N)^a b^N \bmod N^2$ for random $a$ and $b$ from $\mathbb{Z}_N^*$. Compute $\theta = a\beta\lambda \bmod N$ for a random $\beta \in \mathbb{Z}_N^*$ where $\lambda = \text{lcm}(p-1, q-1)$ is the Carmichael number for $N$. Let $(N, g, \theta)$ and $\lambda$ be the public and private keys, respectively . Use the Asmuth-Bloom SSS to share $\beta\lambda$ with $m_0 = N\lambda$.

(2) Let $c = g^w r^N \bmod N^2$ be the ciphertext to be decrypted for some random $r \in \mathbb{Z}_N^*$ where $w$ is the message from $\mathbb{Z}_N$. Assume a coalition $\mathcal{S}$ of size $t$ wants to obtain the message $w = \frac{L(c^{\beta\lambda} \bmod N^2)}{\theta} \bmod N$. We call $s = c^{\beta\lambda} \bmod N^2$ as the *decryptor*. The $i$th user in the coalition knows $m_j$ for all $j \in \mathcal{S}$ and $y_i = y \bmod m_i$ as its secret share.

(3) Each user $i \in \mathcal{S}$ computes

$$u_i = y_i M'_{\mathcal{S},i} M_{\mathcal{S}\setminus\{i\}} \bmod M_{\mathcal{S}}, \tag{29}$$
$$s_i = c^{u_i} \bmod N^2, \tag{30}$$
$$\theta_i = g^{u_i} \bmod N^2. \tag{31}$$

(4) The *incomplete decryptor* $\overline{s}$ is obtained by combining the $s_i$ values

$$\overline{s} = \prod_{i \in \mathcal{S}} s_i \bmod N^2. \tag{32}$$

(5) The $\theta_i$ values will be used to find the exponent which corrects the incomplete decryptor. Compute the incomplete $\overline{\theta}$ as

$$\overline{\theta} = \prod_{i \in \mathcal{S}} \theta_i \bmod N^2. \tag{33}$$

Let $\kappa_s = {c_1}^{M_{\mathcal{S}}} \bmod N^2$ and $\kappa_\theta = g^{-M_{\mathcal{S}}} \bmod N^2$ be the *correctors* for $s$ and $\theta$, respectively. The corrector exponent $\delta$ can be obtained by trying

$$\theta \stackrel{?}{=} L(\overline{\theta}\kappa_\beta^j) \bmod N^2 \tag{34}$$

for $0 \leq j < t$.

(6) Compute the message $w$ as

$$s = \overline{s}{\kappa_s}^{\delta} \bmod N^2, \tag{35}$$

$$w = \frac{L(s)}{\theta} \bmod N. \tag{36}$$

where $\delta$ denotes the value for $j$ that satisfies (34).

The decryptor $\overline{s}$ is *incomplete* and to find the corrector exponent we used a similar approach. When the equality in (34) holds we know that $\theta = a\beta\lambda \bmod N^2$ is the correct value. Also, this equality must hold for one $j$ value, denoted by $\delta$, in the given interval. Actually, in (33) and (34), our purpose is not computing $\theta$ since it is already known. We want to find the corrector exponent $\delta$ to obtain $s$, which is also equal to the one we used to obtain $\theta$.

### *7.1 Security Analysis*

Here, we will prove that the threshold Paillier encryption scheme is semantically secure against under the static adversary model provided that the Paillier encryption scheme is also semantically secure.

**Theorem 7.1** *If the threshold Paillier encryption scheme is not semantically secure against a static adversary who controls exactly $t-1$ users, the standard Paillier scheme is not semantically secure.*

**Proof** The structure of the proof is similar to the one that we used for the previous threshold schemes.

Let $\mathcal{S}'$ denote the set of users controlled by the adversary. To simulate adversary's view, a random $y$ value is chosen from $\mathbb{Z}_M$, $M = \prod_{i=1}^{t} m_i$. Then, the compromised shares are computed by $y_j = y \bmod m_j$ for $j \in \mathcal{S}'$. Note that, in the threshold Paillier signature scheme, $y$ can be any integer in $\mathbb{Z}_M$ so the distribution of the shares of corrupted users is indistinguishable from the distribution of the shares in the real case.

Since we have a $(t, n)$-threshold scheme, when we determine the $y_j$ values for $j \in \mathcal{S}'$, the shares of other users are also determined. Although they cannot be computed easily, given a valid message-ciphertext pair $(w, c)$ the decryptor share $s_i$ and $\theta_i$ for a user $i \notin \mathcal{S}'$ can be obtained by

$$s_i = (1 + w\theta N){\kappa_s}^{-\delta_{\mathcal{S}}} \prod_{j \in \mathcal{S}'} ({c_1}^{u_j})^{-1} \bmod N^2, \tag{37}$$

$$\theta_i = (1 + \theta N){\kappa_\theta}^{-\delta_{\mathcal{S}}} \prod_{j \in \mathcal{S}'} (\theta^{u_j})^{-1} \bmod N^2. \tag{38}$$

where $\mathcal{S} = \mathcal{S}' \cup \{i\}$, $\kappa_s = c^{-M_{\mathcal{S}}} \bmod N^2$, $\kappa_\theta = g^{-M_{\mathcal{S}}} \bmod N^2$ and $\delta_{\mathcal{S}}$ is equal to either $\lfloor \frac{\sum_{j \in \mathcal{S}'} u_j}{M_{\mathcal{S}}} \rfloor + 1$ or $\lfloor \frac{\sum_{j \in \mathcal{S}'} u_j}{M_{\mathcal{S}}} \rfloor$. We use the same ideas to choose the value of $\delta_{\mathcal{S}}$ as in the previous simulator so we skip the details and the analysis for the secrecy of the private key in the proof.

Consequently, the output of the simulator is indistinguishable from the adversary's point of view, and hence we proved that the threshold Paillier scheme must be semantically secure if the standard one is. □

## 8 Conclusion and Discussion of the Proposed Schemes

In this paper, sharing of the RSA signature, ElGamal and Paillier decryption functions with the Asmuth-Bloom SSS is investigated. Previous solutions for sharing these functions were typically based on the Shamir SSS [2,8,7,11,13,19] and in one occasion, the Blakley SSS was used for ElGamal decryption [8]. To the best of our knowledge, the schemes described in this paper are the first provably secure FSSs that use the Asmuth-Bloom SSS.

Computational complexity of the proposed schemes is comparable to the ones of earlier proposals. In a straightforward implementation of the threshold RSA scheme, each user needs to do $t + 1$ multiplications, one inversion, and one exponentiation for computing a partial result, which is comparable to the earlier schemes and in fact better than most of them [7,11,19]. Combining the partial results takes $t - 1$ multiplications, plus a correction phase which takes two exponentiations and upto $t + 1$ multiplications.

Computing the partial results in ElGamal and Paillier schemes needs one more exponentiation than computing them in the RSA scheme. For combining the partial results and the correction phase we need $3t - 2$ multiplications, two exponentiations and one inversion in the worst case. For threshold Paillier scheme, this last phase may also require $t$ evaluations of the $L$ function.


### Acknowledgments

We would like to thank İsmail Güloğlu for informative discussions and his comments on this paper. We also give our thanks to Zahir Tezcan for his comments on ElGamal threshold scheme, to Baha Güçlü Dündar and Said Kalkan for their comments on Paillier threshold scheme.

## References


[1] M. Abdalla, O. Chevassut, P.-A. Fouque and D. Pointcheval . A simple threshold authenticated key exchange from short secrets. *Proc. of ASIACRYPT 2005, LNCS 3778*, pages 566–584, 2005.

[2] A. Lysyanskaya and C. Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. *Proc. of ASIACRYPT 2001, LNCS 2248*, pages 331–350, 2001.

[3] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Trans. Information Theory*, 29(2): pages 208–210, 1983.

[4] O. Baudron, P.-A. Fouque, D. Pointcheval, G. Poupard, and J. Stern. Practical multi-candidate election system. *Proc. of PODC 2001, 20th ACM Symposium on Principles of Distributed Computing*, pages 274–283, 2001.

[5] M. Bellare and P. Rogaway. Optimal asymmetric encryption. *Proc. of EUROCRYPT 1994, LNCS 950*, pages 92–111, 1994.

[6] G. Blakley. Safeguarding cryptographic keys. In *Proc. of AFIPS National Computer Conference*, 1979.

[7] Y. Desmedt. Some recent research aspects of threshold cryptography. In *Information Security, First International Workshop ISW '97, LNCS 1196*, pages 158–173 , 1997.

[8] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Proc. of Crypto'89, LNCS 435*, pages 307–315. Springer-Verlag, 1990.

[9] Y. Desmedt and Y. Frankel. Homomorphic zero-knowledge threshold schemes over any finite abelian group. *SIAM Journal on Discrete Mathematics*, 7(4): pages 667–679, 1994.

[10] R. Gennaro, S.Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. *Inf. Comput.*, 164(1): pages 54–84, 2001.

[11] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely? In *Proc. of STOC94*, pages 522–533, 1994.

[12] T. ElGamal A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4): pages 469–472, 1985.

[13] P. A. Fouque, G. Poupard and J. Stern. Sharing decryption in the context of voting or lotteries. *Proc. of FC 2000, 4th International Conference on Financial Cryptography, LNCS 1962*, pages 90–104, 2001.

[14] P. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. *International Journal of Information Security*, 2(34): pages 218-239, 2004.

[15] R. Ostrovsky and W Skeith. Private searching on streaming data. *Proc. of CRYPTO 2005, LNCS 3621*, pages 223–240, 2005.

[16] P. Paillier. Public key cryptosystems based on composite degree residuosity classes *Proc. of EUROCRYPT 1999, LNCS 1592*, pages 223–238, 1999.

[17] R. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Comm. ACM*, 21(2): pages 120-126, 1978.

[18] A. Shamir. How to share a secret? *Comm. ACM*, 22(11): pages 612–613, 1979.

[19] V. Shoup. Practical threshold signatures. *Proc. of EUROCRYPT 2000, LNCS 1807*, pages 207–220, 2000.