Threshold Cryptography Based on Asmuth-Bloom Secret Sharing*

Kamer Kaya** , Ali Aydın Selçuk, Zahir Tezcan

Department of Computer Engineering
Bilkent University
Ankara, 06800, Turkey
{kamer,selcuk,zahir}@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 two novel function sharing schemes, one for the RSA signature and the other for the ElGamal decryption functions, based on the Asmuth-Bloom scheme. To the best of our knowledge, these are the first threshold cryptosystems realized using the Asmuth-Bloom secret sharing. The proposed schemes compare favorably to the earlier function sharing schemes in performance as well as in certain theoretical aspects.

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 [8] based on polynomial interpolation, Blakley [2] based on hyperplane geometry, and Asmuth-Bloom [1] based on the Chinese Remainder Theorem.

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 secret sharing [1, 2, 8] and function sharing [4, 3, 5, 7, 9] have been proposed in the literature. Nearly all existing solutions for function sharing have been based on the Shamir SSS [8].

In this paper, we show how sharing of cryptographic functions can be achieved using the Asmuth-Bloom secret sharing scheme. We give two novel FSSs, one

* This work is supported in part by the Turkish Scientific and Technological Research Agency (TÜBİTAK), under grant number EEEAG-105E065.

** Supported by the Turkish Scientific and Technological Research Agency (TÜBİTAK) Ph.D. scholarship.

for the RSA signature and the other for the ElGamal decryption functions, both based on the Asmuth-Bloom SSS. The proposed schemes, to the best of our knowledge, 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. In Section 3, we discuss the Asmuth-Bloom SSS in detail. After describing the proposed FSSs in Section 4, the paper is concluded with an assessment of the proposed schemes in Section 5.

2 Background

In this section, we give an overview of the field of threshold cryptography and discuss briefly some of the main secret and function sharing schemes in the literature.

2.1 Secret Sharing Schemes

The problem of secret sharing and the first solutions to it were introduced independently by Shamir [8] and Blakley [2] 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, a 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 [8] 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 [2]. 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 any smaller coalition cannot.

A fundamentally different SSS is the scheme of Asmuth and Bloom [1], 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. [4] in 1989. 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 and ElGamal private key operations have been proposed in the literature [4, 3, 5, 6, 7, 9]. Almost all of these schemes are based on the Shamir SSS, with the only exception of one scheme in [4] 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, albeit a relatively less efficient one, was proposed by Desmedt and Frankel [3], 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. [7]. 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 [9] where he removed the need of taking inverses in Lagrangian interpolation altogether by a slight modification in the RSA signature function.

To the best of our knowledge, so far no 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
  $$\prod_{i=1}^{t} m_i > m_0 \prod_{i=1}^{t-1} m_{n-i+1}. \quad (1)$$
  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

In this section, we present two novel FSSs based on the Asmuth-Bloom SSS for sharing the RSA signature and ElGamal decryption functions.

In the original Asmuth-Bloom SSS, the authors proposed a recursive process to solve the system $y \equiv y_i \pmod{m_i}$. Here, we give 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

$$d = y \bmod m_0.$$

As a separate point, note that m_0 in the Asmuth-Bloom SSS 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.

4.1 Sharing of the RSA Signature Function

The following is a FSS based on the Asmuth-Bloom SSS for the RSA signature function:

1. In the 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 message to be signed and assume a coalition $\mathcal{S}$ of size t wants to obtain the signature $s = w^d \bmod N$. The i^{th} person 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

$$u_i = y_i M'_{\mathcal{S},i} M_{\mathcal{S}\setminus\{i\}} \bmod M_{\mathcal{S}}, \tag{3}$$

$$s_i = w^{u_i} \bmod N. \tag{4}$$

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

$$\overline{s} = \prod_{i\in\mathcal{S}} s_i \bmod N. \tag{5}$$

5. Let $\lambda = w^{-M_S} \bmod N$ be the *corrector*. The partial signature can be corrected by trying

$$(\overline{s}\lambda^j)^e = \overline{s}^e(\lambda^e)^j \stackrel{?}{\equiv} w \pmod{N} \tag{6}$$

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

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

where δ denotes the value of j that satisfies (6).

We call the signature $\overline{s}$ generated in (5) *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 (6) 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 (5) will give $d + am_0 + \delta M_{\mathcal{S}}$ in the exponent for some $\delta \leq t-1$. Thus in (5), we obtained

$$\overline{s} = w^{d+\delta M_S} \bmod N \equiv sw^{\delta M_S} \bmod N \equiv s\lambda^{-\delta} \bmod N$$

and for $j = \delta$, equation (6) 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 $s = \overline{s}\lambda^j$ which satisfies (6).

4.2 Sharing of the ElGamal Decryption Function

The following is a FSS based on the Asmuth-Bloom SSS for the ElGamal decryption function:

1. In ElGamal setup, choose $p = 2q + 1$ where q is a large random prime and let g be a generator of $\mathbb{Z}_p^*$. Let $\alpha \in \{1, \ldots, p-1\}$ and $\beta = g^\alpha \bmod p$ be the private and the public key, respectively. Use Asmuth-Bloom SSS for sharing the private key α 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 plaintext message. The coalition $\mathcal{S}$ of t users wants to obtain the plaintext $w = sc_2 \bmod p$ for $s = (c_1^\alpha)^{-1} \bmod p$. The i^{th} person 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

$$u_i = y_i M'_{\mathcal{S},i} M_{\mathcal{S}\setminus\{i\}} \bmod M_{\mathcal{S}}, \tag{7}$$
$$s_i = {c_1}^{-u_i} \bmod p, \tag{8}$$
$$\beta_i = g^{u_i} \bmod p. \tag{9}$$

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

$$\overline{s} = \prod_{i \in \mathcal{S}} s_i \bmod p. \tag{10}$$

5. The β_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

$$\overline{\beta} = \prod_{i \in \mathcal{S}} \beta_i \bmod p. \tag{11}$$

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

$$\overline{\beta}\lambda_\beta^j \stackrel{?}{\equiv} \beta \pmod p \tag{12}$$

for $0 \leq j < t$.

6. Compute the plaintext message w as

$$s = \overline{s}{\lambda_s}^\delta \bmod p, \tag{13}$$

$$w = sc_2 \bmod p. \tag{14}$$

where δ denotes the value for j that satisfies (12).

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 (12) holds we know that $\beta = g^\alpha \bmod p$ is the correct public key. This equality must hold for one j value, denoted by δ, in the given interval because since the u_i values in (7) and (9) are first reduced in modulo $M_\mathcal{S}$. So, combining t of them will give $\alpha + am_0 + \delta M_\mathcal{S}$ in the exponent in (11) for some $\delta \leq t - 1$. Thus in (11), we obtained

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

and for $j = \delta$ equality must hold. Actually, in (11) and (12), our purpose is not computing the public key since it is already known. We want to find the corrector exponent δ to obtain s, which is also equal to one we use to obtain β. The equality can be verified as seen below:

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

5 Discussion of the Proposed Schemes

In this paper, sharing of the RSA signature and ElGamal decryption functions with the Asmuth-Bloom SSS is investigated. Previous solutions for sharing these functions were typically based on the Shamir SSS [4, 3, 7, 9] and in one occasion, the Blakley SSS was used for ElGamal decryption [4]. To the best of our knowledge, the schemes described in this paper are the first that use the Asmuth-Bloom SSS for function sharing.

Computational complexity of the proposed schemes also compare quite favorably to the earlier proposals. In a straightforward implementation, 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 [4, 3, 7, 9]. Combining the partial results takes $t-1$ multiplications, plus possibly a correction phase which takes an exponentiation and $t-1$ multiplications.

Acknowledgments

We would like to thank İsmail Güloğlu for some very informative discussions and his comments on this paper.

References

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

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

[3] Y. Desmedt. Some recent research aspects of threshold cryptography. In *Information Security, First International Workshop ISW '97*, number 1196 in Lecture Notes in Computer Science, pages 158–173. Springer-Verlag, 1997.

[4] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Proc. of Crypto'89*, number 435 in Lecture Notes in Computer Science, pages 307–315. Springer-Verlag, 1990.

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

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

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

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

[9] V. Shoup. Practical threshold signatures. In *Proc. of EUROCRYPT 2000, Lecture Notes in Computer Science, LNCS 1807*, pages 207–220, 2000.