On automatic secret generation and sharing : part I

KAMIL KULESZA, ZBIGNIEW KOTULSKI
Institute of Fundamental Technological Research, Polish Academy of Sciences
ul.Świętokrzyska 21, 00-049, Warsaw Poland, e-mail: {kkulesza, zkotulsk}@ippt.gov.pl

Abstract: The secret considered is a binary string of fixed length. In the paper we propose a method of automatic sharing of a known secret. In this case the dealer does not know the secret and the secret's owner does not know the shares. We discuss how to use extended capabilities in the proposed method.

Key words: cryptography, secret sharing, data security, extended key verification protocol

1. INTRODUCTION

Everybody knows situations, where permission to trigger certain action requires approval of several selected persons. Equally important is that any other set of people cannot trigger the action.

Secret sharing allows to splitt a secret into different pieces, called shares, which are given to the participants, such that only certain group (authorized set of participants) can recover the secret. Secret sharing schemes (SSS) were independently invented by George Blakley [2] and Adi Shamir [10]. Many schemes have been presented since, for instance, Asmuth and Bloom [1], Brickell [5], Karin-Greene-Hellman (KGH method) [6]. In our paper we concentrate on the last method.

In KGH the secret is a vector of η numbers $S_\eta = \{s_1, s_2, ..., s_\eta\}$. Any modulus k is chosen, such that $k > \max(s_1, s_2, ..., s_\eta)$. All t participants are given shares that are η-dimensional vectors $S_\eta^{(j)}$, $j = 1,2,...,t$ with elements in Z_k. To retrieve the secret they have to add the vectors component-wise in Z_k.

For $k = 2$, KGH method works like $\oplus$ (XOR) on η-bits numbers, much in the same way like Vernam one-time pad. If t participants are needed to recover the secret, adding $t-1$ (or less) shares reveals no information about secret itself.

Once secret sharing was introduced, people started to develop extended capabilities. Some of examples are: detection of cheaters (e.g. [8],[9]), multi-secret threshold schemes (e.g., [8]), pre-positioned secret sharing schemes (e.g., [8]).

Anonymous and random secret sharing was studied by Blundo, Giorgio Gaggia, Stinson in [3], [4]. Some of ideas in automatic secret sharing and generation come from the same field.

Dealer of the secret is the entity that assigns secret shares to the participants. Usually, the dealer has to know the secret in order to share it. This gives dealer advantage over ordinary secret participants. There are situations, where such advantage can lead to abuse.

Automatic sharing of a known secret addresses problem of secret owner not trusting the dealer. Using such a method owner can easily share the secret. The resulting secret shares are random. It may have added feature, that even secret owner knows neither secret shares, nor their distribution. The later decreases chances of owner interfering with the shared secret.

The paper consists of two parts with the following outline:
Part I: preliminaries are given in section 2; we also state useful property of binary vectors' set. Next section brings algorithms for automatic sharing for the existing secret. Proposed methods support extended capabilities, which apart from being interesting theoretical constructs on their own, allow greater flexibility in the applications of secret sharing schemes. Part II: we present method for automatic secret generation and sharing, next we discuss further research for results from both parts of the paper. Methods presented in both parts form automatic secret generation and sharing (ASGS).

Remarks about procedures and algorithms presented in this paper. Every routine is described in three parts:

a. Informal description. It states the purpose of routine, describes what is being done and specifies output (when needed). Such description should be enough to comprehend the paper and get main idea behind presented methods.
b. Routines written in pseudocode, resembling high level programming language (say C++). Level of detail is much higher than in description part. Reading through pseudocode might be tedious, but rewarding in the sense that allows appreciate proposed routines in full extend.
c. Discussion (if needed). Methods and results are formally justified.

2. PRELIMINARIES

In order to formally present procedures and algorithms, one needs to introduce notation. Further, we describe two devices and their functions. First comes random number generator; its output strings have good statistical properties (e.g., see [7]). Next comes the accumulator, which is a dumb, automatic device that memory cannot be accessed otherwise than by predefined functions. Its embedded capabilities are described below. In further considerations m_i denotes *l*-bit vector. Given set A, its cardinality (number of elements) is denoted by $|A|$.

RAND yields m_i obtained from a random number generator.

ACC denotes the value of l-bit memory register. Register's functions are:
ACC.reset sets all bits in the memory register to 0,
ACC.read yields *ACC*,

ACC.store(x) yields $ACC = ACC \oplus x$ (performs bitwise XOR of *ACC* with the input binary vector *x*, result is stored to *ACC*).

Accumulator consists of l-bit memory register together with defined above functions. It has also some storage capacity separate from memory register. Accumulator can execute functions and operations as described in procedures.

Secure communication channel. In this paper we assume that all the communication between protocol parties is done in the way that only communicating parties know plaintext. Whenever we use command like "send", we presume that no third party can know the message contents. There is extensive literature on this subject, interested reader can for instance consult [8].

Encapsulation. Entities and devices taking part in the protocol can exchange information with others only via interface. Inner state of the entity (e.g. contents of memory registers) is hidden (encapsulated) and remains unknown for external observers.

The idea of automatic secret generation and sharing is based on the following property of binary vectors.

Basic property: Let $m_i, i = 1,2,...,n$, such that

$$\bigoplus_{i=1}^{n} m_i = \vec{0}, \tag{1}$$

form the set M. For any partition of M into two disjoined subsets C_1, C_2, that is such that $C_1 \cup C_2 = M$, $C_1 \cap C_2 = \varnothing$, holds:

$$\bigoplus_{m_i \in C_1} m_i = \bigoplus_{m_i \in C_2} m_i . \quad \blacksquare \tag{2}$$

Now we present the procedure that generates set of binary vectors M.

Procedure description: *GenerateM* creates set of n binary vectors m_i, satisfying condition (1). Procedure is carried out by the Accumulator.

Procedure 1: ***GenerateM(*** n ***)***

Accumulator:

ACC.reset;
  for $i = 1$ to $n-1$ do
    m_i := *RAND*
    ACC.store (m_i)
    save m_i
  end //for
$m_n = ACC.read$
save m_n
return $M = \{m_1 \quad m_2 \quad ... \quad m_n\}$
end // *GenerateM*

Discussion: We claim that the generated set *M* satisfies condition (1). First, statistically independent random vectors $m_i, i = 1,2,...,n-1$ are generated, while $m_n = \bigoplus_{i=1}^{n-1} m_i$, so $\bigoplus_{i=1}^{n} m_i = \left(\bigoplus_{i=1}^{n-1} m_i\right) \oplus m_n = \left(\bigoplus_{i=1}^{n-1} m_i\right) \oplus \left(\bigoplus_{i=1}^{n-1} m_i\right) = \vec{0}$. ■

3. AUTOMATIC SECRET SHARING

To share secret S, secret owner has to generate set $S^{(o)} = \{s_1^{(o)} \quad s_2^{(o)} \quad ... \quad s_n^{(o)}\}$, such $\bigoplus_{i=1}^{n} s_i^{(o)} = S$.

Automatic secret sharing algorithm takes away responsibility, for proper construction of the secret shares, from the owner. Algorithm *FastShare* provides an automatic tool to complete this task. Next, comes algorithm *SaveShares* that adds up two more capabilities:

a. Shares are prepared using secret mask provided by an external dealer.
b. Owner knows neither distributed shares, nor their assignment to the participants. Once the shares are distributed by *SaveShares*, they have to be activated by the algorithm *ActivateShares*.

Finally, we discuss how automatic secret sharing can be used to implement secret sharing schemes with extended capabilities (e.g. see [8]).

3.1 Known secret sharing

FastShare is the tool that provides fast and automatic sharing for a known secret.

Algorithm description: *FastShare* takes secret S and n (number of secret participants). Accumulator generates random $s_i^{(o)}, i = 1,2,...,n-1$. Every $s_i^{(o)}$ is added to the ACC and simultaneously saved. To obtain $s_n^{(o)}$ the secret *S* is added to ACC. Next, ACC value is read and saved as $s_n^{(o)}$. Algorithm returns $S^{(o)} = \{s_1^{(o)} \quad s_2^{(o)} \quad ... \quad s_n^{(o)}\}$.

Algorithm 1:*FastShare*(S , n)

Accumulator:
ACC.reset
for $i = 1$ to $n-1$
$s_i^{(o)} := RAND$
ACC.store($s_i^{(o)}$)
save $s_i^{(o)}$
end// for
Owner: Send secret to Accumulator

Accumulator:
ACC.store(S) //adding secret to the accumulator
$s_n^{(o)} := ACC.read$
save $s_n^{(o)}$
return $S^{(o)} = \left\{ s_1^{(o)} \quad s_2^{(o)} \quad ... \quad s_n^{(o)} \right\}$
end//*FastShare*

Discussion:

1. We claim that *FastShare* produces random secret shares, due to the fact that all of them originate from a random number generator. First $n-1$ shares are purely random, while the last one results from bitwise XOR of the secret and random number. More formally, $s_n^{(o)} = \bigoplus_{i=1}^{n-1} s_i^{(o)} \oplus S$. So, $s_n^{(o)}$ is random.

2. All secret shares combine to S. Just observe: $\bigoplus_{i=1}^{n} s_i^{(o)} = \bigoplus_{i=1}^{n-1} s_i^{(o)} \oplus s_n^{(o)} = \bigoplus_{i=1}^{n-1} s_i^{(o)} \oplus \left(\bigoplus_{i=1}^{n-1} s_i^{(o)} \oplus S \right) = S$. ■

3.2 Confidential secret sharing

We present two algorithms. First, algorithm *SaveShares* will be described, algorithm *ActivateShares* follows. *SaveShares* shares secret S using secret sharing mask M provided by dealer. In the method the following conditions hold:

a. Dealer does not know S;
b. Secret owner does not know M;
c. Secret owner does not know secret shares and their assignment to the secret participants

Algorithm description: *SaveShares* requires cooperation of two parties: Dealer and Owner. First, Dealer uses *GenerateM* to create secret sharing mask M, such that $\bigoplus_{m_i \in M} m_i = \vec{0}$. He also creates set K of encryption keys k_i, such that $\bigoplus_{k_i \in M} k_i \neq \vec{0}$. M elements are encrypted using corresponding keys from K to form encrypted mask set C. $\begin{bmatrix} m_1 \\ m_2 \\ : \\ m_n \end{bmatrix} \oplus \begin{bmatrix} k_1 \\ k_2 \\ : \\ k_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ : \\ c_n \end{bmatrix}$ or $M \oplus K = C$

Dealer stores K and sends C to the Owner. Owner shares original secret S using *FastShare* to obtain $S^{(o)}$. Using C and $S^{(o)}$ he obtains $S^{(p)}$, which elements are randomly distributed to the participants. $\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \oplus \begin{bmatrix} s_1^{(o)} \\ s_2^{(o)} \\ \vdots \\ s_n^{(o)} \end{bmatrix} = \begin{bmatrix} s_1^{(p)} \\ s_2^{(p)} \\ \vdots \\ s_n^{(p)} \end{bmatrix}$ or $C \oplus S^{(o)} = S^{(p)}$

Participants receive secret shares from $S^{(p)}$ and store them.

Algorithm 2: ***SaveShares***

Dealer:
GenerateM(n)
ACC.reset
Owner:
FastShare(S, n)
for $i = 1$ to n
Dealer:
Label < k_i generation>:
$k_i := RAND$
ACC.store(k_i*)*
if ($i == n$ AND *ACC.read* $== \vec{0}$) {
ACC.store(k_i*)* //remove k_i from *ACC*
go to < k_i generation> // generate k_i again
} // end if
save k_i
$c_i := m_i \oplus k_i$
send c_i to Owner
Owner: $s_i^{(p)} := c_i \oplus s_i^{(o)}$
send $s_i^{(p)}$ to randomly chosen P_j [1]
Participant P_j:
$s_j^{(p)} := s_i^{(p)}$ // share index *i* is updated
save $s_j^{(p)}$ // participant stores his secret share[2]

[1] $j \in \{1,2,...,n\}$, one participant is allowed to obtain only one secret share. Once $s_i^{(o)}$ is send to particular P_j, this participant is removed from the set of participants eligible to obtain secret share.

[2] Secret share $s_i^{(p)}$ that was sent to the participant P_j has now the same index j as the participant.

end//for
end//*SaveShares*

Discussion: Note, that $\bigoplus_{i=1}^{n} s_i^{(p)} \neq S$. So, all secret participants, upon combining their shares, will not receive S. The rest of discussion is postponed after Algorithm 3. ■

ActivateShares is used to activate secret shares that were distributed to secret participants using *SaveShares*.

Algorithm description: *ActivateShares* requires cooperation of two parties: Dealer and Owner (of the secret). Dealer contacts participant P_1. Once participant's identity is established participant obtains one key from the set K. Participant combines k_i with $s_i^{(p)}$ to obtain activated share $s_i^{(a)}$. Action is repeated for all participants.

The algorithm yields $S^{(a)} = S^{(p)} \oplus K$, where $S^{(a)} = \{s_1^{(a)} \quad s_2^{(a)} \quad \ldots \quad s_n^{(a)}\}$.

Algorithm 3:*ActivateShares*

for $i = 1$ to n
Dealer:
contacts P_i
starts identification procedure
if ($identification == 1$) sends k_i to P_i
Participant P_i:
$s_i^{(a)} := s_i^{(p)} + k_i$
saves $s_i^{(a)}$ // activated share is stored
end//for
end//*ActivateShares*

Discussion: Once secret shares are activated, S can be recovered by standard KGH procedure. We claim that $\bigoplus_{i=1}^{n} s_i^{(a)} = S$. To see it one has to combine results from two algorithms *SaveShares* and *ActivateShares* :

$$\bigoplus_{i=1}^{n} s_i^{(a)} = \bigoplus_{i=1}^{n} \left(s_i^{(p)} \oplus k_i^{(d)}\right) = \bigoplus_{i=1}^{n} \left(c_i \oplus s_i^{(o)} \oplus k_i^{(d)}\right) =$$

$$= \bigoplus_{i=1}^{n} \left(m_i \oplus k_i^{(p)} \oplus s_i^{(o)} \oplus k_i^{(d)}\right) = \bigoplus_{i=1}^{n} \left(m_i \oplus s_i^{(o)} \oplus k_i^{(p)} \oplus k_i^{(d)}\right)$$

One should note that particular participant P_i usually obtains two different keys from K. Key $k_i^{(p)}$ comes from the Owner embedded in $s_i^{(p)}$, while $k_i^{(d)}$ comes from the Dealer as a part of *ActivateShares*. Hence, $\bigoplus_{i=1}^{n} \left(k_i^{(p)} \oplus k_i^{(d)}\right) = K \oplus K = \vec{0}$ and $\bigoplus_{i=1}^{n} s_i^{(a)} = \bigoplus_{i=1}^{n} \left(m_i \oplus s_i^{(o)}\right) = \bigoplus_{i=1}^{n} s_i^{(o)} = S$, since $\bigoplus_{i=1}^{n} m_i = \vec{0}$ ■

3.3 Remarks

1. To create single authorized set of participants both algorithms have to be executed. Hence, to obtain many authorized sets of participants, multiple execution of *SaveShares* and *ActivateShares* take place.

2. Extended capabilities. Algorithms defined above can be easily adapted to enable pre-positioned secret sharing. In [8] pre-positioned secret sharing schemes are described as that: „All necessary secret information is put in place excepting a single (constant) share which must later be communicated, e.g., by broadcast, to activate the scheme." In order to implement this capability in our case it is enough to separate execution of *SaveShares* from *ActivateShares*. Scheme is initialised by *SaveShares.* When the time comes, it is activated by using *ActivateShares.* In addition, algorithm *ActivateShares* can be modified, so it will send key values only to selected secret participants. For instance, assume that only one participant is selected. To activate the scheme he obtains $\bigoplus_{k_i \in M} k_i$ as the key. Another possible modification can lead towards public initialization. In this case value of $\bigoplus_{k_i \in M} k_i$ is made public by algorithm *ActivateShares*, so secret participants make use of it to recover original secret.

3. Security discussion. Automatic secret sharing security is based on KGH security (see [6]), combined with encapsulation and use of secure communication channels. We consider method secure, although strict proof of security has not carried out yet.

4. ASGS is further discussed in the part II of this paper.

4. REFERENCES

[1] Asmuth C. and Bloom J. 1983. 'A modular approach to key safeguarding'. *IEEE Transactions on Information Theory* IT-29, pp. 208-211

[2] Blakley G.R. 1979. 'Safeguarding cryptographic keys'. *Proceedings AFIPS 1979 National Computer Conference,* pp. 313-317.

[3] Blundo C., Giorgio Gaggia A., Stinson D.R. 1997.'On the dealer's randomness required in secret sharing schemes'. *Designs, Codes and Cryptography* 11, pp. 107-122.

[4] Blundo C., Stinson D.R. 1997.' Anonymous secret sharing schemes'. *Discrete Applied Mathematics* 77, pp. 13-28.

[5] Brickell E.F. 1989. 'Some ideal secret sharing schemes' *Journal of Combinatorial Mathematics and Combinatorial Computing* 6, pp. 105-113.

[6] Karnin E.D., J.W. Greene, and Hellman M.E. 1983. 'On secret sharing systems'. *IEEE Transactions on Information Theory* IT-29, pp. 35-41.

[7] Kotulski Z. 2001. 'Random number generators: algorithms, testing, applications'. (Polish) *Mat. Stosow.* No. 2(43), pp. 32-66.

[8] Menezes A.J, van Oorschot P. and Vanstone S.C. 1997. '*Handbook of Applied Cryptography*'. CRC Press, Boca Raton.

[9] Pieprzyk J. 1995. '*An introduction to cryptography*'. draft from the Internet.

[10] Shamir A. 1979. 'How to share a secret'. *Communication of the ACM* 22, pp. 612-613.