On automatic secret generation and sharing for Karin-Greene-Hellman scheme

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 secret generation and sharing. We show how to simultaneously generate and share random secret. Such a secret remains unknown until it is reconstructed. Next, 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 split 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 [11]. 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. [9],[10]), multi-secret threshold schemes (e.g., [9]), pre-positioned secret sharing schemes (e.g., [9]).

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 secret generation and sharing (ASGS) allows computing and sharing the secret "on the spot", when it is not predefined. This is typical situation, that secret helps to identify authorized set of participants upon recovery. In such an application any element from certain set (say, all *l*-bit vectors) can be a secret. Automatic secret generation allows random generation of the secret and elimination of the secret Owner. First feature is important even without elimination of the secret Owner. It makes the secret choice "owner independent"; hence decrease chances for the Owner related attack. For instance: users in computer systems have strong inclination to use as the passwords character strings, that have some meaning for them. The most popular choices are spouse/kids names and cars' registration numbers.

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 outline of the paper is the following:

Preliminaries are given in section 2; we also state basic property of binary vectors' set. Next section brings description of methods for automatic secret generation and sharing. In section 4 outline of method for automatic sharing for the existing secret is presented. 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.

Finally, we make few remarks about procedures and algorithms presented in this paper. Short description is provided for every routine. 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. In selected cases (volume constrains) we give 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. Pseudocode for all presented routines can be found in preliminary version of this paper [8].

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 [9].

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

$$\bigoplus_{i=1}^{n} m_i = \vec{0}, \qquad (1)$$

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

$$\bigoplus_{m_i \in C_1} m_i = \bigoplus_{m_i \in C_2} m_i . \quad \blacksquare \qquad (2)$$

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 $M = \{m_1, \quad m_2, \quad \dots \quad , m_n\}$
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 GENERATION AND SHARING

Longman's "Dictionary of Contemporary English" describes secret as "something kept hidden or known only to a few people". Still, there are few basic questions about nature of the secret, that need to be answered:

- When does the secret existence begin?
- Can secret exist before it is created?
- Can secret existence be described by binary variable?
- Can secret exist unknown to anyone; do we need at least one secret holder?
- If secret is shared, how one can verify its validity upon combining the shares?

In our approach secret existence begins, when it is generated. However, for the secret that is generated in the form of distributed shares, moment of creation comes when shares are combined for the first time. Before that moment, secret exists only in some potential (virtual) state. Nobody knows the secret, though secret shares exist, because they have never been combined. In order to assemble it, cooperation of authorized set of participants is required.

In such a situation, there are only two ways to recover secret: by guess or by cooperation of participants from the authorized set. The first way can be feasibly controlled by the size of the secret space, while the other one is the legitimate secret recovery procedure.

In case of the KGH (see [6]) secret sharing scheme, the process of creating secret shares destroys original copy of the secret. Once shares are combined, the secret is recovered. Recovered secret has to be checked against original secret in order to validate it. Hence, there must exist primary (template) copy of the secret. This can be seen from different perspective: recovered secret allows to identify and validate authorized set of participants, so, the template copy is required for comparison. For instance, consider opening bank vault. One copy of the secret is shared between bank employees that can open vault (the authorized set of secret participants). Second copy is programmed into the opening mechanism. When the employees input their combined shares, it can check whether they recover proper secret.

We propose the mechanisms, that allow automatic secret generation, such that:

a. The generated secret attains a randomly generated value.
b. Two copies of the secret are created.
c. Both secret copies are created in a distributed form.
d. Nobody knows the secret till the shares from the authorized set are combined.
e. Distributed secret shares can be replicated without compromising the secret.
f. The secret shares resulting from replication have different values then the source shares.

3.1 Basic secret shares generation

In this section we present algorithm that creates a secret simultaneously in two distributed copies.

Let $s_i^{(1)}$ and $s_i^{(2)}$ be some secret shares in KGH secret sharing scheme, let S denote the secret shared. Now, we show how to generate two authorized set of secret shares $U^{(1)} = \{s_1^{(1)}, s_2^{(1)}, \ldots, s_d^{(1)}\}$ and $U^{(2)} = \{s_1^{(2)}, s_2^{(2)}, \ldots, s_n^{(2)}\}$, such that $\bigoplus_{i \in U^{(1)}} s_i^{(1)} = S = \bigoplus_{i \in U^{(2)}} s_i^{(2)}$. $U^{(1)}$ is authorized set of primary secret shares that is used for verification of $U^{(2)}$. $U^{(2)}$ is called authorized set of user secret shares or, for the reasons that will become clear later, authorized set of master secret shares. To generate $U^{(1)}$ and $U^{(2)}$ algorithm *SetGenerateM* is used.

Algorithm description: *SetGenerateM* creates $U^{(1)}$ and $U^{(2)}$, such that $|U^{(1)}| = d$, $|U^{(2)}| = n$. First, *GenerateM* is used to create set M, such $|M| = d + n$. Next, M is partitioned into $U^{(1)}$ and $U^{(2)}$. The Accumulator executes algorithm automatically.

Algorithm 1: ***SetGenerateM(*** d ***,*** n ***)***

```
Accumulator:
    GenerateM( $d + n$ )
        for $i = 1$ to $d$ do // preparing $U^{(1)}$
            $s_i^{(1)} := m_i$
            save $s_i^{(1)}$
        end //for
        for $i = d + 1$ to $d + n$ do // preparing $U^{(2)}$
            $j := i - d$
            $s_j^{(2)} := m_i$
            save $s_j^{(2)}$
        end //for
    **return** $U^{(1)} = \{s_1^{(1)}, s_2^{(1)}, ..., s_d^{(1)}\}$, $U^{(2)} = \{s_1^{(2)}, s_2^{(2)}, ..., s_n^{(2)}\}$
end// *SetGenerateM*
```

■

So far, generation of secret sets $U^{(1)}$ and $U^{(2)}$, was described. In order to make use of the secret shares they should be distributed to secret shares participants. Shares distribution is carried out via secure communication channel. Little modification (using send instead of save) of *SetGenerateM* allows distribution of shares once they are created. Due to the volume constrains this topic will be omitted. Usually, one participant from the authorized set is assigned one secret share. Let $P_i^{(n)}$ denote secret share participant that was assigned the share $s_i^{(n)}$ from $U^{(n)}$. When $|U^{(1)}| = 1$, one is dealing with degenerate case, where $s_1^{(1)} = S$. It is noteworthy that, when $|U^{(1)}| > 1$, shares assignment to different participants $P_i^{(1)}$ allows to introduce extended capabilities in the secret sharing scheme. One of instances could be split control over secret verification procedure.

3.2 Secret shares replication

Algorithm 1 allows only two sets of secret shares to be created. Usually, only$U^{(2)}$ will be available for secret participants, while $U^{(1)}$ is reserved for shares verification. Often, it is required that there are more than one authorized sets of participants. On the other hand property used in Algorithm 1 does not allow creating more than two authorized sets. The problem is: how to share the secret further without recovering it's value?

This question can be answered by distributed replication of $U^{(2)}$ into $U^{(3)}$. Although all participants $P_i^{(2)}$ take part in the replication, they do not disclose information allowing secret recovery. Any of $P_i^{(2)}$ should obtain no information about any of $s_i^{(3)}$. Writing these properties formally:

1. $\bigoplus_{s_i^{(2)} \in U^{(2)}} s_i^{(2)} = \bigoplus_{s_i^{(3)} \in U^{(3)}} s_i^{(3)} = S$.
2. $P_i^{(2)}$ knows nothing about any of $s_i^{(3)}$.

Such approach does not compromise S and allows to maintain all previously discussed automatic secret generation and sharing features.

3.2.1 Authorized set replication (same cardinality sets)

The authorized set satisfies: $\left|U^{(2)}\right| = \left|U^{(3)}\right| = n$, $U^{(2)} = \left\{s_1^{(2)}, s_2^{(2)}, ..., s_n^{(2)}\right\}$, $U^{(3)} = \left\{s_1^{(3)}, s_2^{(3)}, ..., s_n^{(3)}\right\}$. Procedure *SetReplicate* replicates set $U^{(2)}$ into the set $U^{(3)}$.

Procedure description: *SetReplicate* takes $U^{(2)}$, M, such that $|M| = 2 * \left|U^{(2)}\right|$. First, all participants $P_i^{(2)}$ are assigned corresponding vectors m_i. Each of them performs bitwise XOR on their secret shares and corresponding m_i. Operation result is sent to the Accumulator. Accumulator adds m_{i+n} to form $s_i^{(3)}$, which later is sent to $P_i^{(3)}$. As the result, simultaneous creation and distribution of $U^{(3)}$ takes place.

Procedure 2:*SetReplicate(*M *,*$U^{(2)}$*)*

```
    Accumulator:
    $n := \left|U^{(2)}\right|$
        for $i = 1$ to $n$
            send $m_i$ to $P_i^{(2)}$
            $P_i^{(2)}$: $\omega_i^{(2)} := s_i^{(2)} \oplus m_i$ // $\omega$ is l-bit vector (local variable)
        end//for
        for $i = 1$ to $n$
            $P_i^{(2)}$ send $\omega_i^{(2)}$ to Accumulator
            Accumulator: $s_i^{(3)} := \omega_i^{(2)} \oplus m_{i+n}$
            send $s_i^{(3)}$ to $P_i^{(3)}$
        end// for
end//SetReplicate
```

■

Algorithm *EqualSetReplicate* is the final result in this section.

Algorithm description: *EqualSetReplicate* takes $U^{(2)}$. It uses *SetReplicate* to create and distribute set $U^{(3)}$, such $\left|U^{(2)}\right| = \left|U^{(3)}\right| = n$.

Algorithm 2: ***EqualSetReplicate(***$U^{(2)}$***)***

```
    Accumulator:
    $n := \left|U^{(2)}\right|$
    $M := GenerateM(2n)$
    SetReplicate($M$ ,$U^{(2)}$)
end// EqualSetReplicate
```

Discussion: We claim that *EqualSetReplicate* fulfils requirements stated at the beginning of section 3:

1. $\bigoplus_{i=1}^{n} s_i^{(3)} = \bigoplus_{i=1}^{n} \left(s_i^{(2)} \oplus m_i \oplus m_{i+n}\right) = \left(\bigoplus_{i=1}^{n} s_i^{(2)}\right) \oplus \left(\bigoplus_{i=1}^{2n} m_i\right) = \bigoplus_{i=1}^{n} s_i^{(2)}$ as requested.
2. All $s_i^{(3)}$ result from XOR of some elements from $U^{(2)}$ with random m_i, m_{i+n} hence they are random numbers. ■

3.2.2 Authorized set replication (different cardinality sets)

For $|U_2| \neq |U_3|$ there are two possibilities:

Case 1: The authorized set satisfies: $n < d$, $U^{(2)} = \{s_1^{(2)}, s_2^{(2)}, ..., s_n^{(2)}\}$, $U^{(3)} = \{s_1^{(3)}, s_2^{(3)}, ..., s_d^{(3)}\}$. The algorithm *SetReplicateToBigger* takes$U^{(2)}$. It uses SetReplicate to create and distribute set $U^{(3)}$, such $n = |U_2| < |U_3| = d$.

Algorithm 3 description: *SetReplicateToBigger* takes d and $U^{(2)}$. It generates M , such that $|M| = d$. Next, it uses *SetReplicate* to create and distribute first n elements from $U^{(3)}$. As the result participants $P_i^{(3)}$ for $i \leq n$ have their secret shares assigned. Remaining participants $P_i^{(3)}$ are assigned m_i ($i > n$) not used by *SetReplicate*. As the result $U^{(3)}$, such $n = |U_2| < |U_3| = d$ is created and distributed. ■

Case 2: The authorized set satisfies: $n > d$, $U^{(2)} = \{s_1^{(2)}, s_2^{(2)}, ..., s_n^{(2)}\}$, $U^{(3)} = \{s_1^{(3)}, s_2^{(3)}, ..., s_d^{(3)}\}$. The algorithm *SetReplicateToSmaller* takes$U^{(2)}$. It uses SetReplicate to create and distribute set $U^{(3)}$, such $n = |U_2| > |U_3| = d$.

Algorithm 4 description: *SetReplicateToSmaller* takes d and $U^{(2)}$. It generates M , such that $|M| = n + d - 1$. Next, it uses *SetReplicate* code to create n secret shares $s_i^{(3)}$. First $d-1$ shares are sent to corresponding participants $P_i^{(3)}$. Remaining $s_i^{(3)}$ ($i \in \{d, d+1, ..., n\}$) are combined to form $s_d^{(3)}$ that is sent to $P_d^{(3)}$. As the result $U^{(3)}$, such that $n = |U_2| > |U_3| = d$ is created and distributed. ■

3.3 Remarks

1. All three algorithms meet requirements stated at the beginning of the paper. Combing this fact with security proof for KGH secret sharing scheme [6], encapsulation and use of secure communication channels, enables us to consider them as secure. Certainly, detailed proofs of security are yet to be constructed.
2. To obtain many authorized sets of participants, multiple replication of $U^{(2)}$ takes place. In such instance $U^{(2)}$ is used as the master copy (template) for all $U^{(n)}$, $n \geq 3$. For this reason it is called authorized set of master secret shares.

3. Proposed algorithms can accommodate arbitrary access structure, when combined with cumulative array construction (e.g. see [10]).

4. AUTOMATIC SECRET SHARING

To share secret S, secret Owner has to generate set $S^{(o)} = \left\{ s_1^{(o)}, \ s_2^{(o)}, \ \dots \ , s_n^{(o)} \right\}$, 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 [9]).

4.1 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 $s_i^{(o)}, i = 1,2,\dots,n-1$. 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 $S^{(o)} = \left\{ s_1^{(o)}, \ s_2^{(o)}, \ \dots \ , s_n^{(o)} \right\}$.

Algorithm 5:*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 $S^{(o)} = \{s_1^{(o)},\ s_2^{(o)},\ \ldots\ , s_n^{(o)}\}$
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:

$$\bigoplus_{i=1}^{n} s_i^{(o)} = \bigoplus_{i=1}^{n-1} s_i^{(o)} \oplus s_n^{(o)} = \bigoplus_{i=1}^{n-1} s_i^{(o)} \oplus \left(\bigoplus_{i=1}^{n-1} s_i^{(o)} \oplus S \right) = S\ .\blacksquare$$

4.2 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 6 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. ■

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

Algorithm 7 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 $S^{(a)} = \left\{ s_1^{(a)} \quad s_2^{(a)} \quad \ldots \quad s_n^{(a)} \right\}$. ■

4.3 Remarks

1. Security discussion is analogous as in the section 3.3 .
2. 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.
3. Extended capabilities. Algorithms defined above can be easily adapted to enable pre-positioned secret sharing. In [9] 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 initialized 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.

5. FURTHER RESEARCH

We hope that in both parts of the paper we managed to present in comprehensible way all basic algorithms for ASGS. However, much work still needs to be done. Major research tasks are:

1. Generalization into arbitrary access structures. This seems to be relatively simple task nevertheless it requires proper formalization.
2. Adding more extended capabilities to both methods. Some of possibilities were outlined in the paper. Set of extra functions that can be embedded into secret sharing scheme is much bigger. Authors are busy working in this field.
3. Construction of security proofs.

6. AKNOWLEDGMENT

The paper was partially supported by State Committee for Scientific Research, grant no.8 T11D 020 19. The authors wish to thank our friends: Kris Gaj, Karol Górski and Konrad Kulesza for discussion and useful comments.

7. 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] Kulesza K., Kotulski Z. 2002. ‘On automatic secret generation and sharing, Part 1,2’ *Proceedings of the 9th International Conference on Advanced Computer Systems*, ACS’2002, pp. 81-96.

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

[10] Pieprzyk J. 1995. ‘*An introduction to cryptography*’. draft from the Internet.

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