ON SECRET SHARING SCHEMES WITH EXTENDED CAPABILITIES

Kamil Kulesza, Zbigniew Kotulski
Institute of Fundamental Technological Research
Polish Academy of Sciences
ul.Świętokrzyska 21, 00-049 Warsaw, Poland
Email: kkulesza, zkotulsk@ippt.gov.pl

ABSTRACT

Secret sharing allows split control over the secret. Secret parts known as secret shares are distributed to different shareholders (locations). Secret can be recovered upon combination of sufficient number of shares. Research into theoretical foundations, development of widespread applications, as well as, new attacks resulted in great need for extra features of existing schemes.
We describe extended capabilities for secret sharing schemes. List of discussed issues contains (but is not limited to): multiple and different size sets of shareholders, schemes activation, multi-secret schemes, cheating detection, secret shares renewal/maintaince, missing shares recovery. Next, research results into extended capabilities for Karin-Green-Hellman scheme are presented. We concentrate on Proactive Secret Sharing (PSS) and Publicly Verifiable Secret Sharing (PVSS). Results are in the form of collections of algorithm in pseudocode, that can be easily implemented.
Finally, we discuss trade-offs between secret security and extended capabilities.

Keywords: cryptography, secret sharing, data security, extended capabilities

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. Just to bring few instances:
1. Bank transactions. When customer wants to draw lump sum from the account, his order has countersigned by 2 out of 3 authorized bank employees. Order validated by such procedure can be further processed.
2. Corporate governance. In order to issue valid decision in the name of the Board of Directors, threshold number of members of the Board has to approve decision.
3. Military. Crucial operations, for instance launch of ballistic missile, can be executed only when initialization code is entered. Secret sharing allows split control over such code, resulting in split control over initiation of crucial operations.

Secret sharing allows splitting a secret into different pieces, called shares, which are given to the participants. The secret is usually shared and distributed to the participants by the dealer. Only certain group (authorized set of participants) can recover the secret. Secret Sharing Schemes (SSS) were independently invented by George Blakley [1] and Adi Shamir [2]. Many schemes have been presented since, for instance, Asmuth and Bloom [3], Brickell [4], Karin-Greene-Hellman (KGH method) [5]. Majority of given above schemes are so-called (t,n) threshold schemes, that any t (but not less) participants out of total number of n participants can recover the secret. Later some of the schemes were generalized to arbitrary collections of authorized sets of participants, known as general access structure. We will describe it in greater detail in the next section.

Formal foundation of secret sharing was formulated using the information theory. Two important concepts were defined: ideal and perfect schemes. Informally speaking, scheme is ideal if each share has the same length as the secret. Ideal property can be thought as measure of efficiency of secret sharing scheme. We consider schemes to be perfect, if any non-authorized set of participants, cannot deduce extra information about the secret from their shares, compared to people not having shares at all. One can think about perfectness property as measure of security of secret distribution. Interested reader can consult, for instance [6] for more formal definitions.

Research into theory resulted in better understanding of secret sharing, while growing number of practical applications required greater flexibility and functionality. This lead to development of secret sharing schemes with extended capabilities that are the topic of this paper. It consists of 6 sections with the following outline:

1. Introduction. You are reading it now.
2. State of art. It provides overview of secret sharing schemes with extended capabilities and detailed description of KGH scheme.
3. Preliminaries. Useful properties extensively used further in the text, are derived.
4. Proactive Secret Sharing (PSS). Algorithms to implement PSS are described and discussed.
5. Publicly Verifiable Secret Sharing (PVSS). Again algorithms to implement PVSS are described and discussed.
6. Remarks. Concluding remarks are stated.

At the end of introduction, we make some comments on procedures and algorithms presented in this paper. Every routine is described in three consecutive steps:

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 the description part. Reading through pseudocode might be tedious, but rewarding in the sense that allows appreciate proposed routines in full extend. In C++ fashion we use // sign for comments, while in pseudocode.
c. Discussion (if needed). Methods and results are formally justified.

2. STATE OF ART

At the beginning of this section we describe Karin-Green-Hellman (KGH) secret sharing scheme that is extensively used for the rest of the paper. In the second part we provide overview of secret sharing schemes with extended capabilities.

KGH description

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 reveal no information about secret itself, hence KGH is perfect, see [6].

Extended capabilities

Extended capabilities arise from a need for greater flexibility and functionality of the secret sharing scheme. We describe two directions of development:

- various authorized sets of participants,
- added features (extended capabilities in classical sense).

Various authorized sets of participants

1. General access structure. First secret sharing schemes were (t,n) threshold schemes, with $t \le n$. In practice, it is often needed that only certain specified subsets of the participants should be able to recover the secret. Hence, it is not surprising that next stage in development of secret shares came with introduction of general access.

The access structure describes all the authorized subsets. To design the access structure with required capabilities, the cumulative array construction can be used, for details see, for example [6]. Combining cumulative arrays with KGH method, one obtains implementation of general secret sharing scheme (see, e.g., [6]). Although KGH is ideal for (t,t) threshold scheme, it is not always true for an arbitrary access structure. Design of access structures is difficult, interesting results in this field are given in [7].

2. Anonymous secret sharing schemes. In such schemes secret is reconstructed using "blackbox", that separates secret share from its holder. Hence, secret is recovered without knowing neither identities of participants, nor

shares to participants assignment (e.g. [9]).
Added features
Some schemes have added functions that were first called "extended capabilities".
1. Proactive Secret Sharing (PSS). PSS is used for long-lived and sensitive secrets, see [10]. Often nature of the secret does not allow changing it (for instance proprietary trade secrets or legal documents). PSS allows periodically renewing secret shares, leaving the same secret. Method for recovery of missing or corrupted shares should be also available. In addition enrollment/disenrollment of secret participants should be supported. We propose PSS for KGH scheme in the section 4.
2. Verifiable Secret Sharing (VSS) and Publicly Verifiable Secret Sharing (PVSS). VSS allows detecting cheating by secret participants and/or the secret dealer (e.g. [11], [12]). Verification capability is especially important, if secret consistency is crucial (for instance cryptographic master keys).
Cheating can result not only in obstruction of the protocol (for instance by recovering invalid secret), but also may allow dishonest parties to recover secret on their own, see [13]. Verification process requires presence of trusted third party or can be performed directly between parties of the protocol. When it takes place in public or uses publicly available data, we have Publicly Verifiable Secret Sharing, for instance see [14]. We propose PVSS for KGH scheme in the section 5.
3. Automatic Secret Generation and Sharing (ASGS). There are situation, when secret is generated just before sharing. One of them might be "identification mode" of operation for secret sharing. In this situation, participants that are capable to recover the secret, identify themselves as authorized set of participants. The content of the secret is secondary to that capability, and often is generated on the spot.
Automatic secret generation addresses issue of generation and distribution of such secret. Often secret is generated in the distributed form and remains unknown till the time it is recovered for the first time.
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 allows secret owner to eliminate the dealer and share the secret automatically. 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.
An instance of ASGS is given in [15] and [16].
4. Pre-positioned secret sharing schemes.
Such schemes allow the dealer to keep control over secret recovery process (e.g. its time). 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."
5. Multi-secret threshold schemes. Menezes, van Oorschot and Vanstone in [8] wrote:" In these secret sharing schemes different secrets are associated with different authorized subsets".
After this little survey it can be though that extended capabilities are quite well described. Nevertheless, still there is a room for refinement and more extensions. For example: extended capabilities over general access structure give rise to many new problems. Often using extension yields some price to pay, for instance, usually verification capabilities result in situation, where scheme losses perfectness property. Its security is based on some computationally difficult problems like discreet logarithm (e.g., see [14]), hence resulting in theoretically weaker security class of secret sharing scheme.
Because requirements for various extended capabilities are often contrary to each other, building secret sharing schemes with combined extended capabilities is a very challenging problem.

3. PRELIMINARIES

In order to formally present procedures and algorithms, one needs to introduce notation. We describe two devices and their functions. First comes random number generator; its output strings have good statistical properties (e.g., see [17]). 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 (no. 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 the message 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.

Let's denote:

S as the whole secret

P_i as ith participant

s_i as secret share assigned to the participant P_i

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, that is such that $C_1 \cup C_2 = M, C_1 \cap C_2 = \varnothing$, holds:

$$\bigoplus_{m_i \in C_1} m_i = \bigoplus_{m_i \in C_2} m_i. \qquad ■$$

(2)

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

Procedure description:

GenerateM creates the set of n binary vectors m_i, satisfying condition (1). The Accumulator carries out procedure.

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 $\bigoplus_{i=1}^{n} m_i = \left(\bigoplus_{i=1}^{n-1} m_i\right) \oplus m_n = \left(\bigoplus_{i=1}^{n-1} m_i\right) \oplus \left(\bigoplus_{i=1}^{n-1} m_i\right) = \vec{0}$

■

4. PROACTIVE SECRET SHARING

We propose efficient proactive secret sharing for KGH scheme. It allows to:

a. periodically renew secret shares without changing the secret,

b. enroll/disenroll participants

c. recover missing/corrupted secret shares

Let us start from the first capability.

RenewShares allows renewing secret shares in the way such that the renewed shares combine to the same secret as original ones.

Algorithm description:

RenewShares uses *GenerateM* to obtain set of n binary vectors that next are pairwise XORed with existing shares by the secret participants.

Each of the secret participants stores result of operation as the renewed share.

Algorithm 1: *RenewShares(*n*)*

```
Dealer:
GenerateM($n$):
    for $i = 1$ to $n$
        send $m_i$ to $P_i$
        Participant $P_i$:
        $s_i := s_i \oplus m_i$
        save $s_i$
    end// for
end//RenewShares
```

Discussion:
1. We claim that *RenewShares* produces random secret shares, due to the fact that all of them result from XOR of random vector provided by *GenerateM* and secret share.
2. All renewed secret shares combine to S. Just observe:

$$\bigoplus_{i=1}^{n} s_i = \bigoplus_{i=1}^{n} (s_i \oplus m_i)$$

$$= \bigoplus_{i=1}^{n} s_i \oplus \bigoplus_{i=1}^{n} m_i = \bigoplus_{i=1}^{n} s_i \oplus \vec{0} = S$$

3. Multiple repetitions of the algorithm do not reveal any additional information, hence
it can be used for periodical renewal of the shares. ■

EnrollParticipants allows creating new authorized set of participants by adding new participants. Already existing secret shares are modified in the way that all participants from new authorized set are needed in order to recover the secret.

Algorithm description:
EnrollParticipants uses *GenerateM* to obtain a set of $n+h$ binary vectors, where h denotes the number of participants to be added.
First n vectors are pairwise XORed with existing shares by the secret participants. Each new participant is assigned one of remaining binary vectors.
All the secret participants store result of operation as their secret share.

Algorithm 2: *EnrollParticipants(*h*)*

```
Dealer:
GenerateM($n+h$):
    for $i = 1$ to $n$
        send $m_i$ to $P_i$
        Participant $P_i$:
        $s_i := s_i \oplus m_i$
        save $s_i$
    end// for
    for $i = n+1$ to $n+h$
        send $m_i$ to $P_i$
        Participant $P_i$:
        $s_i := m_i$
        save $s_i$
    end// for
end//EnrollParticipants
```

Discussion:
1. We claim that *EnrollParticipants* produces random secret shares. Some of them are random vectors generated by *GenerateM*, while the others result from XOR of random vectors and existing secret shares.
2. All renewed secret shares combine to S. Just observe:

$$\bigoplus_{i=1}^{n+h} s_i = \bigoplus_{i=1}^{n} (s_i \oplus m_i) \oplus \bigoplus_{i=n+1}^{n+h} m_i =$$

$$\bigoplus_{i=1}^{n} s_i \oplus \bigoplus_{i=1}^{n+h} m_i = \bigoplus_{i=1}^{n} s_i \oplus \vec{0} = S$$

■

DisenrollParticipants allows removing participants from the authorized set and creating new smaller set. Already existing secret shares are modified in the way that only participants from new authorized set are needed to recover the secret.

Algorithm description:
DisenrollParticipants takes the list of participants to be removed ($\overline{P} = \{P_1^{(d)}, ..., P_l^{(d)}\}$) as the parameter. First, dealer has to obtain and combine all secret shares from $\overline{P}$. Combined shares are stored in the Accumulator. Next, she uses *GenerateM* to obtain set of $n-h$ binary vectors, where h denotes the number of participants to be removed. First generated vector is XORed with the combined shares of removed participants (the Accumulator contents).
All binary vectors generated, including modified one are pairwise XORed with the existing shares by the secret participants. Each of the secret participants stores the result of operation as his new secret share.

Algorithm 3: *DisenrollParticipants($\overline{P}$)*

```
Dealer:
 ACC.reset
    for $i = 1$ to $h$
      contacts $P_i^{(d)}$
      Participant $P_i^{(d)}$: sends Dealer $s_i^{(d)}$
      $ACC.store(s_i^{(d)})$
    end// for

 GenerateM($n-h$)
 $m_1 := m_1 \oplus ACC.read$ // $m_1$ XOR all $s_i^{(d)}$

 remaining secret participants are assigned
 new indexes $i \in \{1,...,n-h\}$ to form $s_i^{(n)}$

    for $i = 1$ to $n-h$
       send $m_i$ to $P_i$
          Participant $P_i$:
          $s_i^{(n)} := s_i^{(n)} \oplus m_i$
          save $s_i^{(n)}$
    end// for
end//DisenrollParticipants
```

Discussion:
1. We claim that *DisenrollParticipants* produces random secret shares, same reasoning like in the Algorithm 1 applies.
2. All renewed secret shares combine to S.
Just, observe that once participants are removed: $s_1 = \left(\bigoplus_{i=1}^{h} s_i^{(d)} \otimes m_1\right)$ we obtain:

$$\bigoplus_{i=1}^{n-h} s_i^{(n)} = \left(\bigoplus_{i=1}^{h} s_i^{(d)} \otimes m_1\right) \oplus \bigoplus_{i=2}^{n-h} s_i^{(n)} =$$

$$\bigoplus_{i=1}^{n} s_i \oplus \bigoplus_{i=1}^{n-h} m_i = \bigoplus_{i=1}^{n} s_i \oplus \vec{0} = S$$

3. We require cooperation from secret participants that are to be removed. When there is no such cooperation, they shares need to be recovered. The next algorithm addresses this problem. ■

When secret shares are lost or corrupted, *RecoverShares* is used to obtain value of missing shares. In case of KGH scheme, this has to be performed comparing remaining valid shares with the secret. The difference (XOR) yields value of the missing shares.
One possible approach would be to have a trusted entity (for instance the dealer), which maintains the secret. The other is to use distributed computations. Second one seams to be more secure, because nobody knows the secret. But there is the price to pay; we need second authorized set of participants to perform the comparison.
Let's denote the original authorized set of participants as $P^{(1)} = \{P_1^{(1)}, P_2^{(1)}, ..., P_h^{(1)}\}$ and corresponding set of shares assigned the participants as $U^{(1)} = \{s_1^{(1)}, s_2^{(1)}, ..., s_h^{(1)}\}$. In this set only n out of total h shares are available (remaining shares are missing), hence it can be written as $U^{(1)} = \{s_1^{(1)}, s_2^{(1)}, ..., s_n^{(1)}, s_{n+1}^{(1)}, ..., s_h^{(1)}\}$.
The second authorized set of participants $P^{(2)} = \{P_1^{(2)}, P_2^{(2)}, ..., P_g^{(2)}\}$ is needed for the recovery. Corresponding set of shares assigned to the participants can be written as $U^{(2)} = \{s_1^{(2)}, s_2^{(2)}, ..., s_g^{(2)}\}$.
For both sets of shares it holds, $\bigoplus_{i \in U^{(1)}} s_i^{(1)} = S = \bigoplus_{i \in U^{(2)}} s_i^{(2)}$. It should be emphasized that $U^{(2)}$ has to be complete in order to recover valid missing shares.
Algorithm description:
RecoverShares uses Accumulator to combine all available secret shares from $U^{(1)}$ and $U^{(2)}$. One round: a share from $U^{(1)}$ is sent to the Accumulator, next a share from $U^{(2)}$ is sent. Operation is repeated until there are no shares in both sets. Accumulator contains combined value of missing shares. Algorithm returns this value.

Algorithm 4: *RecoverShares($U^{(1)}$,$U^{(2)}$)*

```
ACC.reset
 If $(n \ge g)$ then $counter := n$
 else $counter := g$
   for $i = 1$ to $counter$
      Participant $P_i^{(1)}$:
      if $(n \ge i)$ send $s_i^{(1)}$ to Accumulator
      else send $s_i^{(1)} = \vec{0}$ to Accumulator
      Accumulator: $ACC.store(s_i^{(1)})$
      Participant $P_i^{(2)}$:
      if $(g \ge i)$ send $s_i^{(2)}$ to Accumulator
      else send $s_i^{(2)} = \vec{0}$ to Accumulator
```

```
        Accumulator: $ACC.store(s_i^{(2)})$
    end// for
    Dealer:
      $RecoverShares := ACC.read$
end//RecoverShares
```

Discussion:

1. Method of sending shares $s_i^{(1)}/s_i^{(2)}$ to the Accumulator enforce security upon the secret. At no point in time, value of the secret cannot be recover using Accumulator contents.
2. We claim, that once all available secret shares are sent to the Accumulator, it contains value of missing shares. This can be demonstrated from $\bigoplus_{i\in U^{(1)}} s_i^{(1)} = S = \bigoplus_{i\in U^{(2)}} s_i^{(2)}$ that yields $\left(\bigoplus_{i\in U^{(1)}} s_i^{(1)}\right) \oplus \left(\bigoplus_{i\in U^{(2)}} s_i^{(2)}\right) = \vec{0}$.

Take $U^{(1)}$ that is partitioned into two subsets $U_a^{(1)}$ and $U_b^{(1)}$, such that $U_a^{(1)} \cup U_b^{(1)} = U^{(1)}$, $U_a^{(1)} \cap U_b^{(1)} = \varnothing$.

If $\left(\bigoplus_{i\in U_a^{(1)}} s_i^{(1)} \oplus \bigoplus_{i\in U_b^{(1)}} s_i^{(1)}\right) \oplus \left(\bigoplus_{i\in U^{(2)}} s_i^{(2)}\right) = \vec{0}$, then $\left(\bigoplus_{i\in U_a^{(1)}} s_i^{(1)}\right) \oplus \left(\bigoplus_{i\in U^{(2)}} s_i^{(2)}\right) = \bigoplus_{i\in U_b^{(1)}} s_i^{(1)}$.

3. Algorithm *RecoverShares* can be used as an extension for *DisenrollParticipants,* when participants that are to be removed do not cooperate. ■

RecoverShares allows us to recover missing shares, that for the sake of further discussion are denoted as *LostShares*. In order to restore the scheme functionality, one would attempt to assign these shares to the participants that lost them. *LostShares* comes in the combined form, that is as the difference (XOR) between combination of available shares and the secret. If more then one share is missing, individual shares have to be computed and distributed.

On the other hand, taking into account that some of the shares from authorized set were corrupted, it is good practice to renew all the shares. *Recover&Renew allows* recovering missing shares and assigning to every participant from the authorized set renewed secret share.

Algorithm description:

Recover&Renew uses algorithm *RecoverShares* to obtain *LostShares*. It also uses *GenerateM* to obtain set of h binary vectors, where h denotes the number of participants in the authorized set. First generated vector is XORed with *LostShares*. All binary vectors generated, including modified one, are sent to the secret participants. If the secret participant has a valid secret share, this share is XORed with received vector to obtain a renewed secret share. Otherwise the renewed secret share is equal to the received vector.

Algorithm 5: *Recover&Renew*

```
  Dealer:
    GenerateM($h$)
      $LostShares := RecoverShares$
        $m_1 := m_1 \oplus LostShares$
          for $i = 1$ to $h$
              send $m_i$ to $P_i^{(1)}$
              Participant $P_i^{(1)}$:
                  if has $P_i^{(1)}$ has valid $s_i^{(1)}$
                      then $s_i^{(1)} := s_i^{(1)} \oplus m_i$
                  else $s_i^{(1)} := m_i$
                  save $s_i^{(1)}$
          end// for
end// Recover&Renew
```

Discussion:

1. We claim that *Recover&Renew* produces random secret shares, same reasoning like in the Algorithm 2 applies.
2. All renewed secret shares combine to S. Applying the same notation, like in the discussion for algorithm *RecoverShares* one may write:

$LostShares := \bigoplus_{i\in U_b^{(1)}} s_i^{(1)}$, while the valid shares are all $s_i^{(1)}$ such that $i \in U_a^{(1)}$. Algorithm produces $s_1^{(1)} := s_1^{(1)} \oplus \left(\bigoplus_{i\in U_b^{(1)}} s_i^{(1)} \oplus m_1\right)$.

XOR of all remaining shares gives:

$$\bigoplus_{i=2}^{h} s_i^{(1)} = \left(\bigoplus_{i\in U_a^{(1)}} s_i^{(1)}\right) \oplus \left(\bigoplus_{i=2}^{h} m_i\right).$$

Finally applying $U_a^{(1)} \cup U_b^{(1)} = U^{(1)}$, yields

$$s_1 \oplus \left(\bigoplus_{i=2}^{h} s_i\right) = \left(\bigoplus_{i\in U^{(1)}} s_i^{(1)}\right) \oplus \left(\bigoplus_{i=1}^{h} m_i\right) = S . ■$$

5. PUBLICLY VERIFIABLE SECRET SHARING (PVSS)

We propose efficient PVSS for KGH scheme that allows checking whether shares of the secret are correctly distributed and protects against cheating dealer. Method provides verification, without compromising the secret. Using it, secret participants in various authorized sets, can verify whether upon combining they will recover the same value of the secret S.

PVSS is collection of algorithms that provide method for public verification of secret shares.

Scheme works for the secret sharing schemes with two or more authorized sets of participants. We present the case with two authorized sets of participants.

First, dealer has to share the secret into two sets $U^{(1)} = \{s_1^{(1)}, s_2^{(1)}, ..., s_h^{(1)}\}$, $U^{(2)} = \{s_1^{(2)}, s_2^{(2)}, ..., s_g^{(2)}\}$, such that $\bigoplus_{i \in U^{(1)}} s_i^{(1)} = S = \bigoplus_{i \in U^{(2)}} s_i^{(2)}$, for instance see [16].

Once the secret is shared into two authorized sets, *DistributeShares&Keys* can be used.

Algorithm description:

DistributeShares&Keys uses *RAND* to obtain random encryption keys for the secret shares. For every secret share, the key value is sent to the corresponding secret participant. The value of the secret share encrypted using derived encryption key is made public. Procedure is performed for both authorized sets of participants ($U^{(1)}, U^{(2)}$).

Algorithm 6: *DistributeShares&Keys*

```
ACC.reset
    for $i = 1$ to $h$
        $k_i^{(1)} := RAND$
        send $k_i^{(1)}$ to $P_i^{(1)}$
          // participant $P_i^{(1)}$ obtains his key
        $c_i^{(1)} = s_i^{(1)} \oplus k_i^{(1)}$
        publish $c_i^{(1)}$
    end// for
    for $i = 1$ to $g$
        $k_i^{(2)} := RAND$
        send $k_i^{(2)}$ to $P_i^{(2)}$
          // participant $P_i^{(2)}$ obtains his key
        $c_i^{(2)} = s_i^{(2)} \oplus k_i^{(2)}$
        publish $c_i^{(2)}$
    end// for
end// DistributeShares&Keys
```

Discussion:

1.When algorithm is completed, the following hold:

a. The encrypted value of secret shares $c_i^{(1)}$, $c_i^{(2)}$ are publicly available for all secret shares from $U^{(1)}, U^{(2)}$,

b. The secret key $k_i^{(1)} / k_i^{(2)}$ is known only to the participant concerned.

2. Participants from the authorized set are able to recover the secret. For instance, consider participants from $U^{(1)}$, encrypted shares are formed as follows

$$\begin{bmatrix} s_1^{(1)} \\ s_2^{(1)} \\ : \\ s_h^{(1)} \end{bmatrix} \oplus \begin{bmatrix} k_1^{(1)} \\ k_2^{(1)} \\ : \\ k_h^{(1)} \end{bmatrix} = \begin{bmatrix} c_1^{(1)} \\ c_2^{(1)} \\ : \\ c_h^{(1)} \end{bmatrix}$$

When it comes to secret recovery, the following situation has place:

$$\begin{bmatrix} c_1^{(1)} \\ c_2^{(1)} \\ : \\ c_h^{(1)} \end{bmatrix} \oplus \begin{bmatrix} k_1^{(1)} \\ k_2^{(1)} \\ : \\ k_h^{(1)} \end{bmatrix} = \begin{bmatrix} s_1^{(1)} \\ s_2^{(1)} \\ : \\ s_h^{(1)} \end{bmatrix}$$ so the secret shares can be computed, allowing combining the secret $\bigoplus_{i \in U^{(1)}} s_i^{(1)} = S$.

When the keys and the encrypted shares are distributed, the secret participants can verify validity of their shares (or check dealer's honesty). The first step is made with the algorithm *RecoverXORedKeys*.

Algorithm description:

RecoverXORedKeys uses Accumulator to combine all secret keys $k_i^{(1)} / k_i^{(2)}$. One round: a key from $U^{(1)}$ is sent to the Accumulator, next a key from $U^{(2)}$ is sent. Operation is repeated until all the keys are in the Accumulator.

Algorithm 7: *RecoverXORedKeys*($U^{(1)}, U^{(2)}$)

```
ACC.reset
  If $(n \geq m)$ then $counter := n$
  else $counter := m$
    for $i = 1$ to $counter$
      Participant $P_i^{(1)}$:
        if $(n \ge i)$ send $s_i^{(1)}$ to Accumulator
        else $s_i = \vec{0}$ to Accumulator
      Accumulator:
        $ACC.store(s_i^{(1)})$
      Participant $P_i^{(2)}$:
        if $(m \ge i)$ send $s_i^{(2)}$ to Accumulator
        else $s_i = \vec{0}$ to Accumulator
      Accumulator:
        $ACC.store(s_i^{(2)})$
    end// for
  RecoverXORedKeys := ACC.read
end// RecoverXORedKeys
```

Discussion:

1. Method of sending keys to the Accumulator enforce security upon value of combined keys from one set. At no point in time, value of the combined keys cannot be recovered using Accumulator contents. ■

Having a way to recover the key, we are ready for algorithm *Verify*.

Algorithm 8: Verify$(U^{(1)}, U^{(2)})$

```
1. Publicly XOR all encrypted secret shares,
   store result in XOREncryptedShares
2. Run RecoverXORedKeys, store result in
   XORedKeys
3. If (XORedKeys== XOREncryptedShares)
   verification POSITIVE
   else verification NEGATIVE
end// Verify
```

Discussion:

1. We claim that if dealer is honest *XORedKeys= XOREncryptedShares.* Note that:

$$\left(\bigoplus_{i \in U^{(1)}} c_i^{(1)}\right) \oplus \left(\bigoplus_{i \in U^{(2)}} c_i^{(2)}\right) =$$

$$\left(\bigoplus_{i \in U^{(1)}} \left(k_i^{(1)} \oplus s_i^{(1)}\right)\right) \oplus \left(\bigoplus_{i \in U^{(2)}} \left(k_i^{(2)} \oplus s_i^{(2)}\right)\right) =$$

$$\left(\bigoplus_{i \in U^{(1)}} k_i^{(1)}\right) \oplus \left(\bigoplus_{i \in U^{(2)}} k_i^{(2)}\right) \oplus \left(\bigoplus_{i \in U^{(1)}} s_i^{(1)}\right) \oplus$$

$$\left(\bigoplus_{i \in U^{(2)}} s_i^{(2)}\right) = \left(\bigoplus_{i \in U^{(1)}} k_i^{(1)}\right) \oplus \left(\bigoplus_{i \in U^{(2)}} k_i^{(2)}\right) \oplus$$

$$S \oplus S = \left(\bigoplus_{i \in U^{(1)}} k_i^{(1)}\right) \oplus \left(\bigoplus_{i \in U^{(2)}} k_i^{(2)}\right)$$

The relation presented above is symmetric in the sense that it does not differentiate between keys and encrypted shares. If dealer attempts to cheat (no matter whether in keys and/or encrypted shares), the equality will not hold. It is satisfied only when both authorized sets of participants receive the same value of the secret.

2.Because no information is revealed about $\bigoplus_{i \in U^{(1)}} k_i^{(1)} / \bigoplus_{i \in U^{(2)}} k_i^{(2)}$ (only they XORed value is provided), hence verification is secure. ■

Algorithm *Verify* is the final result of this section. PVSS, described so far, works for two authorized sets of the secret participants. However, method can be adapted to work for more authorized sets of participants. In such a case, the secret keys in *DistributeShares&Keys* have to be derived for all participants.

Verify can be used in the same form, checking shares validity pairwise. This will require multiple repetitions of the algorithm.

Other possibility arises when number of authorized sets is even. In such a case algorithm *Verify* can be modified, that all authorized set are checked in one round. All secrets S will XOR to $\vec{0}$, because there will be even number of S-es and other properties of algorithm will remain valid.

PVSS can be extended further to detect cheating participants. Major problem is not only to detect, that cheating took place, but also to design protocol in the way that dishonest party will not profit from his actions.

We outline two possibilities:

a. Using two authorized sets of participants. Verification method is similar to the one used to check the dealer's honesty. The two sets recover secret separately using two Accumulators. The contents of the Accumulators is combined in one Accumulator, only when it results in $\vec{0}$, second Accumulator makes its contents available,

b. Using one-way/hash function to protect the secret. A value of such a function is computed using shares sent to participant (e.g., all shares pass via Accumulator, that computes function) and made publicly available. When the secret is recovered Accumulator is used to compute the function. The computed value is compared with one that is publicly available; if they match Accumulator makes its contents available.

6. REMARKS

1. Security discussion. Security for both PSS and PVSS is based on KGH security (see [5]), combined with encapsulation and use of secure communication channels. Shares are random, hence any shares encryption or update does not violate the perfectness property.
 We consider method secure, although strict proof of security is not presented.
2. Proposed schemes allow avoiding traditional trade-off between security and verification possibilities. This comes mainly thanks to properties of KGH scheme. It may require some redundancy (e.g. two authorized sets of participants for secret recovery or verification), but schemes still remain perfect and, if access structure permits, ideal.
3. Combining extended capabilities. Proposed PSS and PVSS allow building KGH based scheme combining both capabilities. It requires sequential execution of both protocols. First shares assigned to participants are verified using PVSS and later maintained by PSS.

AKNOWLEDGMENT

The paper was partially supported by State Committee for Scientific Research, grant no.
8 T11D 020 19.

BIBLIOGRAPHY

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

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

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

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

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

[6] J. Pieprzyk, "An introduction to cryptography", draft from the Internet, 1995.

[7] D.R. Stinson, "Cryptography, Theory and Practice", CRC Press, Boca Raton 1995.

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

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

[10] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive Secret Sharing Or: How to Cope With Perpetual Leakage", Lecture notes in Computer Science, 1996,pp. 339-352 (Advances in Cryptology – CRYPTO'95)

[11] T.P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing", Lecture notes in Computer Science, 1992, pp. 129-140 (Advances in Cryptology – CRYPTO'91)

[12] C. Dwork, "On Verification in Secret Sharing", Lecture notes in Computer Science, 1992, pp. 114-128 (Advances in Cryptology – CRYPTO'91)

[13] M. Tompa, H. Woll, "How to share a secret with cheaters", Journal of Cryptology, 1 (1988), pp. 133-138.

[14] M. Stadler, "Publicly verifiable secret sharing", Lecture notes in Computer Science, 1997,190-199 (Advances in Cryptology – EUROCRYPT'96)

[15] K. Kulesza, Z. Kotulski, "On automatic secret generation and sharing: part I". Proceedings of ACS2002, Szczecin.

[16] K. Kulesza, Z. Kotulski, "On automatic secret generation and sharing: part II". Proceedings of ACS2002, Szczecin

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