On automatic secret generation and sharing: part II

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: In the paper we propose a method of automatic secret generation and sharing. The secret considered is a binary string of fixed length. We show how to simultaneously generate and share random secret. Such a secret remains unknown until it is reconstructed.

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

1. INTRODUCTION

In this part of the paper we use terms and notation introduced in [2], that is the first part of this paper. All preliminaries are also contained there. However, in order to deal with automatic secret generation, we start reasoning from some philosophical background. 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 authorised 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[1] (see [1]) 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.

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. Later 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.

In the following sections 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.

The outline of the paper is as follows: in section 2 we present basic method for automatic secret generation and sharing. Next section brings description of methods for secret replication. Different cardinalities for the replicated sets are being considered. 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.

Again we recall remarks about procedures and algorithms presented in this paper. Every routine is described in three parts:

a. Informal description,
b. Routines written in pseudocode, resembling high level programming language,
c. Discussion. Methods and results are formally justified.

[1] For brief KGH description consult [2]

2. 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 authorised set of secret shares $U^{(1)} = \{s_1^{(1)}, s_2^{(1)}, ..., s_d^{(1)}\}$ and $U^{(2)} = \{s_1^{(2)}, s_2^{(2)}, ..., s_n^{(2)}\}$, such that $\bigoplus_{i \in U^{(1)}} s_i^{(1)} = S = \bigoplus_{i \in U^{(2)}} s_i^{(2)}$. $U^{(1)}$ is authorised set of primary secret shares that is used for verification of $U^{(2)}$. $U^{(2)}$ is called authorised set of user secret shares or, for the reasons that will become clear later, authorised 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* [2] 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

[2] for description of the algorithm *GenerateM* consult [2]

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 $\left|U^{(1)}\right| = 1$, one is dealing with degenerate case, where $s_1^{(1)} = S$. It is noteworthy that, when $\left|U^{(1)}\right| > 1$, 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. 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.

3.1 Authorised set replication (same cardinality sets)

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

Procedure description: *SetReplicate* takes $U^{(2)}$, M, such that $\left|M\right| = 2 * \left|U^{(2)}\right|$. 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 3:*SetReplicate(*M *,*$U^{(2)}$*)*

```
    Accumulator:
    $n := |U^{(2)}|$
        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 $|U^{(2)}| = |U^{(3)}| = n$.

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

```
    Accumulator:
    $n := |U^{(2)}|$
    $M := GenerateM(2n)$
    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 Authorised set replication (different cardinality sets)

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

Case 1: The authorised 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 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.

<u>Algorithm 3:</u> ***SetReplicateToBigger(*** $U^{(2)}$ ***, d)***

```
    $M := GenerateM(d)$
    SetReplicate( $M$ , $U^{(2)}$ ) // assigns shares for participants up to $P_n^{(3)}$
        for $i = n+1$ to $d$
            send $m_i$ to $P_i^{(3)}$
        end//for
end// BiggerSetReplicate
```

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

1. $\bigoplus_{i=1}^{d} s_i^{(3)} = \left\{ \bigoplus_{i=1}^{n} \left(s_i^{(2)} \oplus m_i \right) \right\} \oplus \left(\bigoplus_{i=n+1}^{d} m_i \right) = \left(\bigoplus_{i=1}^{n} s_i^{(2)} \right) \oplus \left(\bigoplus_{i=1}^{d} m_i \right) = \bigoplus_{i=1}^{n} s_i^{(2)}$

2. For $i > n$ all $s_i^{(3)}$ are equal to random numbers m_i. For $i \leq n$ all $s_i^{(3)}$ result from XOR of some elements from $U^{(2)}$ with random m_i, hence are random numbers, too. ■

Case 2: The authorised 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 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.

Algorithm 4: SetReplicateToSmaller$(U^{(2)}, d)$

```
$n := |U^{(2)}|$
$M := GenerateM(n + d - 1)$
    Accumulator:
        for $i = 1$ to $n$
            send $m_i$ to $P_i^{(2)}$
            $P_i^{(2)}$: $\omega_i^{(2)} := s_i^{(2)} \oplus m_i$ // ω is l-bit vector (local variable)
        end//for
        for $i = 1$ to $d - 1$
            $P_i^{(2)}$ 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
    ACC.reset
        for $i = d$ to $n$ // all $\omega_i^{(3)}$ for $i \le d$ were already used
            $P_i^{(2)}$ send $\omega_i^{(2)}$ to Accumulator
            Accumulator: ACC.store( $\omega_i^{(2)}$ )
        end//for
    $s_d^{(3)} = ACC.read$ // $s_d^{(3)} := \bigoplus_{i=l}^{n} \omega_i^{(2)}$
    send $s_d^{(3)}$ to $P_d^{(3)}$
end// SetReplicateToSmaller
```

Discussion: We claim that *SetReplicateToSmaller* fulfils requirements stated at the beginning of section 2:

1. $\bigoplus_{i=1}^{d} s_i^{(3)} = \bigoplus_{i=1}^{d-1} \left(s_i^{(2)} \oplus m_i \oplus m_{i+n}\right) \oplus \left(\bigoplus_{i=d}^{n} \left(s_i^{(2)} \oplus m_i\right)\right) = \bigoplus_{i=1}^{n} s_i^{(2)} \oplus \left(\bigoplus_{i=1}^{n+d-1} m_i \right) = \bigoplus_{i=1}^{n} s_i^{(2)}$

2. All $s_i^{(3)}$ result from XOR of some elements from $U^{(2)}$ with random m_i hence they are random numbers. ■

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 [1], 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 authorised 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 authorised set of master secret shares.

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

4. 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. As stated in the remarks at the end of sections 3 and 4 such a proof needs to be constructed.

5. AKNOWLEDGMENT

The paper (both parts) was partially supported by State Comitte for Scientific Research, grant no.8 T11D 020 19. The autors wish to thank our friends Kris Gaj and Karol Górski for discussion and usefull comments.

6. REFERENCES

[1] 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.

[2] Kulesza K., Kotulski Z. 2002. ‘On automatic secret generation and sharing: part I’. Proceedings of ACS2002, pp. .

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