Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

On ASGS framework: general requirements and an example of implementation*

KULESZA Kamil[1,2], KOTULSKI Zbigniew[2]

([1]*Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge CB3 0WA, UK*)

([2]*Institute of Fundamental Technological Research, Polish Academy of Sciences, Warsaw 00-049, Poland*)

E-mail: K.Kulesza@damtp.cam.ac.uk; Zbigniew.Kotulski@ippt.gov.pl

Received Feb. 2, 2007; revision accepted Feb. 28, 2007

Abstract: In the paper we propose a general, abstract framework for Automatic Secret Generation and Sharing (ASGS) that should be independent of underlying Secret Sharing Scheme (SSS). ASGS allows to prevent the Dealer from knowing the secret. The Basic Property Conjecture (BPC) forms the base of the framework. Due to the level of abstraction, results are portable into the realm of quantum computing.

Two situations are discussed. First concerns simultaneous generation and sharing of the random, prior nonexistent secret. Such a secret remains unknown until it is reconstructed. Next, we propose the framework for automatic sharing of a known secret. In this case the Dealer does not know the secret and the secret Owner does not know the shares. We present opportunities for joining ASGS with other extended capabilities, with special emphasis on PVSS and pre-positioned secret sharing. Finally, we illustrate framework with practical implementation.

Key words: Secret sharing, Security protocols, Dependable systems, Authentication management
doi:10.1631/jzus.2007.A0511 **Document code:** A **CLC number:** TP309

INTRODUCTION

Everybody knows situations, where permission to trigger certain action requires approval of several selected entities. Equally important is that any other set of entities cannot trigger the action. Secret sharing allows a secret to be split into different pieces, called shares, which are given to the participants, such that only certain groups (authorized sets of participants) can recover the secret.

To make this requirement more realistic, one should avoid situations where some of the protocol parties have dominant position. This reasoning resulted in creating the framework for Automatic Secret Generation and Sharing (ASGS).

Secret Sharing Schemes (SSSs) were independently invented by Blakley (1979) and Shamir (1979). Many schemes have been presented ever since, for instance, modular (Asmuth and Bloom, 1983), Brickell (Brickell, 1989), KGH (Karnin *et al*., 1983), discrete-log based threshold cryptosystems (Desmedt and Frankel, 1989). An SSS can operate in two modes:

(1) Split control over the secret. In this case the secret itself is important, hence protected by distributing its pieces to different parties. For instance, it can be applied to control over critical systems and infrastructures like nuclear weapons (Anderson, 2001).

(2) Authentication of the protocol parties. The content of the secret is secondary to the fact that only participants from the authorized set are able to recover it. This property allows to authenticate parties taking part in the protocol. If they are able to recover the valid secret, they are the right ones. The most illustrative popular example comes from spy movies, where two strange people met and they authenticate themselves based on the two halves of the same banknote, with each part in possession of a single person.

* Part of the work was done when the first author was a visiting scholar at DAMTP

Once secret sharing was introduced, people started to develop extended capabilities. Some of examples are: detection of cheaters and secret consistency verification (Stadler, 1996; Menezes *et al*., 1997; Pieprzyk *et al*., 2003), multi-secret threshold schemes (Menezes *et al*., 1997), pre-positioned SSSs (Menezes *et al*., 1997). The other class of extended capabilities focuses on anonymity, randomness and automatization for secret sharing procedures. Anonymous and random secret sharing was studied by Blundo *et al*.(1997) and Blundo and Stinson (1997). Some of ideas in automatic secret sharing and generation originate from the same root.

Although verification capacity can protect against cheating, it usually comes at the price. This fact is related to the paradox stated by David Chaum, that no system can simultaneously provide privacy and integrity. Alternative approach proposed recently (Kulesza *et al*., 2002) seems to be promising shortcut. Nevertheless, the simplest way to stop cheating is to eliminate misbehaving parties from the protocol.

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 participants. There are situations where such an advantage can lead to abuse. For instance, it often happens that secret Dealer is not the secret Owner (e.g., Owner hired the Dealer to share the secret due to the task complexity). In this situation, Owner has to disclose secret to the Dealer. Such a knowledge allows Dealer to make use of the secret without cooperation of the set of authorized participants.

For the first time the problem was discussed in context of discrete-log based threshold cryptosystems by Pedersen (1991). Several papers followed (Li *et al*., 1994; Shoup and Gennaro, 1998) and final solution was presented in (Gennaro *et al*., 1999). The last paper provides secure distributed secret generation for threshold discrete-log based cryptosystem. Analogous solution for the KGH scheme was presented in (Kulesza and Kotulski, 2003). It has added feature of supporting distributed secret generation not only for the threshold scheme, but also for general access structure.

The main contribution of this paper is to propose a generalized, abstract framework that should be independent of underlying SSS. We also want to make it independent of underlying access structure, much like the case presented in (Kulesza and Kotulski, 2003). To discuss the security of proposed framework, we claim that each particular realization of ASGS requires existence of some Basic Property for underlying scheme. We state Basic Property Conjecture (BPC) and discuss its implications. Due to the level of abstraction BPC and resulting framework are portable into the realm of quantum computing.

Having in mind two modes of operation for SSSs, we propose two solutions:

(1) Automatic secret generation and sharing of random, prior nonexistent secret. It allows computing and sharing the secret "on the spot" when it is not predefined. This is typical situation for authentication mode. The secret is generated at random, allowing elimination of the secret Owner.

(2) Automatic sharing of a known secret. The motivation comes from the need to share secret that is fixed and cannot be modified. A good practical example for secret of this kind would be "the secret Coca-Cola formula". An automatic procedure allows the Owner to share the secret. It addresses problem of a secret Owner not trusting the Dealer. It can have an added feature, that even secret Owner knows neither secret shares, nor their distribution. The later decreases chances of Owner interfering with the secret shares.

We present the framework as the collection of procedures and algorithms. Implementation details will vary depending on the characteristic of particular scheme. Hence, in the main part of the paper we provide only functional descriptions. In Appendix B we give example of particular implementation. It allows to fully appreciate interactions between abstract level, represented by the particular realization of Basic Property, and practical implementation for the procedures and algorithms. Suitability for the quantum computing realm comes as an added features. It arises from the fact that underlying scheme and particular realization of the Basic Property are not based on intractable problems.

The article has the following outline: Section 2 is devoted to preliminaries. In Section 3 we state Basic Property Conjecture. The next section contains description of automatic secret generation and sharing of random secret, while Section 5 deals with automatic sharing of known secret. Example of practical implementation for KGH method is given in the appendixes. Appendix A, corresponding to Sections 2

and 3, provides all preliminaries, while Appendix B corresponds to Section 4 of the paper. Preliminary version of these two appendixes appeared in (Kulesza and Kotulski, 2003).

PRELIMINARIES

In order to acquire fundamental understanding of underlying concepts, one needs to start from the philosophical background. Longman's *Dictionary of Contemporary English* describes "secret" as "something kept hidden or known only to a few people". Still, there are basic questions about nature of the "secret", which 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 or is it fuzzy?
- Can secret exist unknown to anyone; do we need at least one secret holder?
- If secret is shared, how can one verify its validity upon combining the shares?
- What does it mean that secret is shared or distributed?

Search for answer to the last question resulted in the development of SSSs. When the first schemes were published, answers for first 4 questions were taken for granted. At that time such an approach was justified, because the goal was to facilitate split control over known secret. The answers were found and formulated in the language of information theory (Karnin *et al*., 1983; Brickell, 1989).

Since ASGS differs from basic SSS, the questions need to be answered again. This is done at the beginning of Sections 4 and 5. Together with described algorithms they result in the general framework for ASGS.

Use of automatic devices

In general framework, we make use of automatic devices and procedures. We favor the approach, that in order to ease analysis and enhance security, they should be as simple as possible (so-called KISS principle). In the paper at least two such devices will be needed:

(1) The random number generator; with output strings having good statistical properties (Knuth, 1997).

(2) The accumulator, which is a dumb, automatic device that memory cannot be accessed otherwise than by predefined functions.

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 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 consult for instance (Menezes *et al*., 1997).

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. Encapsulation, originating in object-oriented paradigm (Budd, 1997), is widely used in various fields of computer science.

Finally, let us provide the notation:

Let K be a secret space, S denotes the secret shared ($S\in K$), while $s_i^{(1)}$ and $s_i^{(2)}$ are secret shares in some SSS.

$C(U)$ denotes combiner algorithm for the given SSS that operates on the authorized set of shares U. $U^{(1)}=\{s_1^{(1)},s_2^{(1)},...,s_d^{(1)}\}$ and $U^{(2)}=\{s_1^{(2)},s_2^{(2)},...,s_n^{(2)}\}$ are two authorized sets of secret shares such that $|U^{(1)}|=d$, $|U^{(2)}|=n$ and $C(U^{(1)})=S=C(U^{(2)})$.

$U^{(1)}$ is called authorized set of primary secret shares that is used for verification of $U^{(2)}$. Set $U^{(2)}$ is called authorized sets of user secret shares or, for the reasons that will become clear later, authorized set of master secret shares.

$P_i^{(n)}$ denotes share participant that was assigned to the secret share $S_i^{(n)}$ from $U^{(n)}$.

SECURITY MODEL DESCRIPTION

The ultimate goal is to build ASGS that will have the same security features as underlying SSS. Speaking in the information theory terms: proposed framework, when applied, should not decrease entropy of S over K (entropy of the secret over the secret space).

Qualitative description of ASGS, which is provided in Sections 4 and 5, allows only stating general

requirements concerning security of the framework. In particular implementations, these points can be expanded into full-blown security proof. Nevertheless, we make few points:

Security of the framework

Security of the framework is based on the use of secure communication channels, simple automatic devices (like Accumulator) and encapsulation principle. All these terms were specified in Section 2. Out of these three, the most unsettled are simple automatic devices. We assume, convincing security proof can be stated, as long as, devices are kept simple. In order to implement ASGS, the Basic Property has to be found for each SSS concerned.

Conjecture 1 (Basic Property Conjecture, BPC) ASGS can be implemented for any SSS that has a property, that there exist operation(s) O with the following characteristics:

(1) O applied to the authorized set of participants (possibly more than one) allows determining consistency of the secret shares (Gennaro *et al*., 1999; Kulesza and Kotulski, 2003);

(2) O will not decrease entropy of S over K;

(3) O can be performed on the shares that are protected by envelopes (e.g., encrypted) (Menezes *et al*., 1997; Pieprzyk *et al*., 2003);

(4) O does not place any restriction on the access structure, except ones resulting from the SSS itself.

BPC states sufficient condition to implement ASGS (described by collection of algorithms described in Sections 4 and 5) for the given SSS.

Examples (of the Basic Property in different SSSs):

(1) Discrete-log based threshold cryptosystems. The Basic Property results from intractability of discrete logarithm. Full description and proof of security is described as DKG (Distributed Key Generation) protocol (Gennaro *et al*., 1999).

(2) KGH scheme. The Basic Property results from properties of bitwise XOR on binary strings. It was described in (Kulesza and Kotulski, 2003). The outline is presented in Appendix A.

(3) Elliptic curves cryptosystems. Two previous examples were describing cases of known SSSs. In this example we adopt a different approach. We start from handy Basic Property and discuss how to build SSS that has ASGS capability. First observe that O can be implemented as a product operation in the abelian group (Herstein, 1964). Consider perfect (Menezes *et al*., 1997) SSS based on elliptic curves (Koblitz, 1993). The simplest implementation of such a scheme would be very much the same like KGH. In such case O will satisfy the Basic Property, provided that shares are embedded into the envelopes.

Remark We do not claim that the only way to build ASGS for the given scheme is to find such Basic Property. Actually, we cannot say anything about possible alternative approach. Yet, we claim that having feasible Basic Property for the given SSS, we can build ASGS for that scheme.

Verification

In ASGS secret shares are derived automatically with help of the simple devices, like Accumulator.

Some of proposed algorithms require interactions between parties of the protocol. Hence, we recommend that shares should be tested for consistency once distributed. The test has to provide information, whether participants from various authorized sets can recover the same secret S. The testing method depends on underlying scheme, but where possible we propose to use some Publicly Verifiable Secret Sharing (PVSS) protocol. It has to support possibility of secure testing of already distributed shares. If Basic Property (as described above) is found, construction of PVSS is possible (Kulesza and Kotulski, 2002).

Remark Capability to build PVSS seems to be related to existence of Basic Property, as stated by BPC. This is definitely the case for KGH scheme in the implementation proposed in (Kulesza and Kotulski, 2002) and for the discrete logarithm based scheme implementation described in (Stadler, 1996).

Security against adversaries with quantum computing power

Second condition of BPC says that O cannot weaken the underlying SSS. In the first example it of BPC means that SSS security is related to computational difficulty of discrete logarithm. Relaying on computational security can have far reaching consequences when quantum paradigm becomes a reality. In this case many intractable problems may become computationally feasible. This in turn would compromise the security of all the systems based on such

problems (Gruska, 1999).

In the second example of BPC SSS, security is not unaffected by introduction of quantum computing capabilities. This construction, whose underlying idea is closely related to the one-time pad, remains provably secure. The point is that defining Basic Property and building ASGS around it allows ordering and simplifying the security discussion for the resulting construct. Once quantum computation is available, in order to examine the ASGS for the particular SSS one has to perform two steps:

(1) Perform security proof of SSS itself;

(2) Check the Basic Property, with special attention paid to the second point of Conjecture 1 (not decreasing entropy).

AUTOMATIC SECRET GENERATION AND SHARING

In this section we discuss automatic secret generation and sharing of random, prior nonexistent secret. First, we provide answers to the questions from Section 2.

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

Once shares are combined, the secret is recovered. Recovered secret has to be checked against the original secret in order to validate it. Hence, there must exist primary (template) copy of the secret. This can be seen from different perspective: authentication mode of operation for SSS should allow to identify and validate authorized set of participants. Hence, 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.

ASGS allows computing and sharing prior nonexistent secret "on the spot". This is typical situation for authentication mode. ASGS allows to prevent the Dealer from knowing the secret or even to eliminate his presence at all. Using proposed procedure, it is also possible to design secret that remains unknown till the time it is recovered. Such secret cannot be compromised in the traditional meaning, because it does not exist until it is recovered. The secret is generated at random. This feature is important even without eliminating the 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 (Anderson, 2001).

ASGS should allow automatic secret generation, such that:

(1) The generated secret is random.

(2) At least two copies of the secret are created. Both secret copies are created in a distributed form.

(3) Nobody knows the secret till the shares from the authorized set are combined.

(4) Distributed secret shares can be replicated without compromising the secret.

(5) Supports the replication of the source set into the target set with different numbers of elements.

(6) The secret shares resulting from replication have different values than the source shares.

(7) ASGS supports the same type of access structure as underlying SSS.

The framework is the collection of the algorithms, whose functional description is provided below. Implementation of each algorithm requires use of the Basic property.

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

Description: *SetGenerateM* is used to generate two distributed copies of the random secret. It produces $U^{(1)}$ and $U^{(2)}$, such that $|U^{(1)}|=d$, $|U^{(2)}|=n$. It is automatically executed by the Accumulator.

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.

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 SSS. One of instances could be split control over secret verification procedure, resulting in pre-positioned SSS (Menezes *et al*., 1997). Algorithm *SetGenerateM* allows only two authorized 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 set of participants. On the other hand, Basic Property often does not allow creating more than two authorized sets (for instance KGH case, see Appendixes A, B). The problem is: how to further share the secret without recovering its 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) $C(U^{(2)})=C(U^{(3)})=S$;

(2) $P_i^{(2)}$ knows nothing about any of $s_i^{(3)}$.

Such approach does not compromise S and allows maintaining all previously introduced ASGS features.

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

Description: *EqualSetReplicate* is used to replicate distributed secret shares into the set with the same number of elements. It uses distributed elements of $U^{(2)}$ to create and distribute set $U^{(3)}$, such $|U^{(2)}|=|U^{(3)}|=n$. It is automatically executed by the Accumulator.

Algorithm 3: *SetReplicateToBigger*($U^{(2)}$, d)

SetReplicateToBigger is used to replicate distributed secret shares into the set with the bigger number of elements. It uses distributed elements of $U^{(2)}$ to create and distribute set $U^{(3)}$, such $n=|U_2|<|U_3|=d$. It is automatically executed by the Accumulator.

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

SetReplicateToSmaller is used to replicate distributed secret shares into the set with the smaller number of elements. It uses distributed elements of $U^{(2)}$ to create and distribute set $U^{(3)}$, such $n=|U_2|>|U_3|=d$. It is automatically executed by the Accumulator.

Remarks

(1) 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.

(2) In ASGS secret shares are derived automatically with the help of simple devices (e.g., Accumulator). Nevertheless we recommend that shares should be tested for consistency once distributed. Namely, whether participants from various authorized set can recover the same secret S.

(3) Above provided algorithms illustrate only basic ideas and provide functional description. Authors are aware that it may look like a "wish list". Reader who is not satisfied with this level of detail is invited to read example of implementation in Appendix B, or consult (Gennaro *et al*., 1999) for the other instance.

AUTOMATIC SECRET SHARING

In this section we discuss the case where the secret is known. It is more classical than that presented in the previous section. Our contribution comes in two parts:

(1) Description of the method allowing automatic sharing of the known secret using simple automatic device;

(2) Using first result, we provide outline of the protocol that allows Owner and Dealer to contribute independently to the process of secret sharing. The protocol may have added feature that both parties know neither secret shares, nor their distribution. The later decreases chances of Owner interfering with the shared secret. Such solution addresses situation where there is no trust between the parties of the protocol.

In addition we require that:

(1) The resulting secret shares are random.

(2) Minimum two copies of the secret have to

exist.

(3) At least one of the copies is in the distributed form.

(4) ASGS supports the same type of access structure as underlying SSS.

To share the secret S, one has to generate set $U^{(o)}=\{s_1^{(o)},s_2^{(o)},...,s_n^{(o)}\}$, such $C(U^{(0)})=S$, where superscript "o" comes from the word "original".

Automatic secret sharing algorithm releases Owner from the responsibility for proper construction of the secret shares. Using such a method Owner can easily share the secret.

Algorithm 5: *FastShare*(S, n)

Description: *FastShare* is the tool that provides fast and automatic sharing for a known secret. Parties of the protocol are Owner and Accumulator. *FastShare* takes from the Owner the secret S and n (the number of secret participants). Algorithm is automatically executed by the Accumulator beyond Owner control. It returns $U^{(o)}=\{s_1^{(o)},s_2^{(o)},...,s_n^{(o)}\}$. Resulting secret shares are not protected against modification by the Owner.

Next algorithm *SafeShares* confidentially shares secret S using secret sharing mask M provided by the Dealer. In the method the following conditions hold:

(1) Dealer does not know S;

(2) Owner does not know M;

(3) Owner does not know secret shares and their assignment to the secret participants.

Algorithm 6: *SafeShares*

Functional description of the algorithm follows, while information flow is illustrated in Fig.1.

Parties of the protocol: Owner, Dealer, Accumulator secret participants.

(1) Dealer prepares mask M needed to share the secret. It can be thought as anonymous envelopes (Menezes *et al*., 1997; Pieprzyk *et al*., 2003) that will be used to hold secret shares. The envelopes are identical and indistinguishable. Their number is equal to the number of secret shares.

(2) Owner shares the secret using *FastShare*, secret shares are placed in the envelopes. The envelopes are placed in the urn. Each secret participant is assigned one randomly chosen envelope. As the result Owner knows neither distributed shares, nor their assignment to the participants.

Secret shares, which are protected by the mask, cannot be combined in order to recover the secret. Once the shares are distributed by *SafeShares*, they have to be activated by the algorithm *ActivateShares*.

Algorithm 7: *ActivateShares*

Functional description of the algorithm follows, while information flow is illustrated in Fig.1.

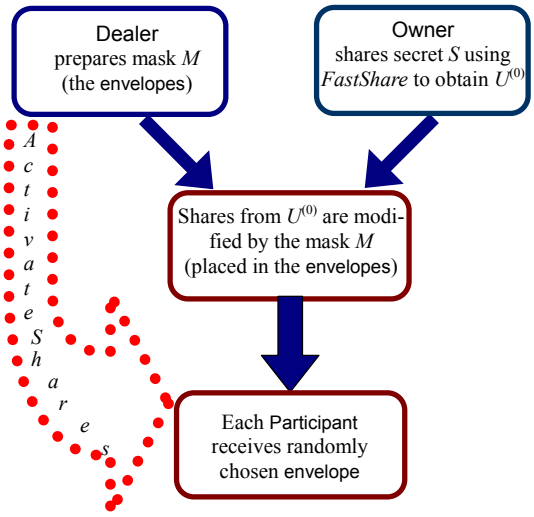

Fig.1 Algorithms *SafeShares* and *ActivateShares*

Parties of the protocol are Dealer and secret participants. Dealer provides secret participants with the information allowing them to remove the mask (extract secret shares from the envelopes). Once procedure is completed shares belonging to the participants from the authorized set can be used to recover the secret.

It is interesting to note, that before secret shares are activated, their existence is only potential. To see it from different perspective: shares cannot be described by binary variable, and they are rather fuzzy. The fuzziness coefficient is given by the probability of proper activation.

Remarks

(1) Multiple authorized sets. To create a single authorized set of participants, both algorithms have to be executed. Hence, to obtain many authorized sets of participants, multiple executions of *SafeShares* and

ActivateShares take place.

(2) Verification. Although much depends on the underlying SSS, proper implementation of *FastShare* should protect against cheating Owner. In particular implementation, unless this fact is proven beyond doubt, we have to assume that Owner can modify the shares. After all, it was one of the reasons for introducing *SafeShares*. At presented level of detail one is not able to discuss cheating possibilities that might be available for the Dealer. Taking this uncertainty into account, we recommend that shares should be tested for consistency once activated. An example of implementation can be found in (Kulesza and Kotulski, 2002).

(3) Extended capabilities. Algorithms defined above can be easily adapted to enable pre-positioned secret sharing (Menezes *et al*., 1997). In order to implement this capability, it is enough to separate execution of *SafeShares* from *ActivateShares*. Hence pre-positioned secret sharing method can have the forms:

(i) The scheme is initialized by *SafeShares*;

(ii) It is activated using *ActivateShares*, when activation time comes.

CONCLUDING REMARKS AND FURTHER RESEARCH

The collection of algorithms forming ASGS framework was provided. Basic Property Conjecture was stated in Section 3. The general security model based on BPC was described and discussed. We also made short trip into the quantum computing realm, showing that framework remains applicable there. Finally, an example of ASGS implementation is provided in the appendixes.

Still much needs to be done. Further research falls into three categories:

(1) Research into ASGS theoretical foundations:

(i) Solid formulation of ASGS framework in terms of the information theory.

(ii) Collecting more facts about Basic Property Conjecture to state it as a theorem. The final result in this field would be proof of such a theorem.

(iii) Finding exact relation between the features of Basic Property and existence of PVSS.

(2) Finding more implementations of ASGS. The problem is related to the question whether every SSS has some form of Basic Property. If the answer is positive, next step is to find such a property and on this foundation build particular ASGS implementation. This also includes formal proofs of security for particular implementations.

(3) Placing ASGS in the broader framework within secret sharing. For instance, joining ASGS with other extended capabilities. In the paper we discussed joining ASGS with pre-positioned secret sharing and PVSS. These two issues can be investigated further. Other natural extension is to use ASGS mechanisms in pro-active secret sharing. One particular implementation is provided in (Kulesza and Kotulski, 2002). Yet, problem requires more general and abstract formulation.

References

Anderson, R., 2001. Security Engineering—A Guide to Building Dependable Distributed Systems. John Wiley & Sons, New York.

Asmuth, C., Bloom, J., 1983. A modular approach to key safeguarding. *IEEE Trans. Inf. Theory*, **29**(2):208-211. [doi:10.1109/TIT.1983.1056651]

Blakley, G.R., 1979. Safeguarding Cryptographic Keys. Proceedings AFIPS 1979 National Computer Conference, p.313-317.

Blundo, C., Stinson, D.R., 1997. Anonymous Secret Sharing Schemes. *Discrete Applied Mathematics*, **77**(1):13-28. [doi:10.1016/S0166-218X(97)89208-6]

Blundo, C., Giorgio Gaggia, A., Stinson, D.R., 1997. On the dealer's randomness required in secret sharing schemes. *Designs, Codes and Cryptography*, **11**(2):107-122. [doi:10.1023/A:1008216403325]

Brickell, E.F., 1989. Some ideal secret sharing schemes. *J. Combin. Math. Combin. Comput.*, **6**:105-113.

Budd, T., 1997. The Introduction to Object-Oriented Programming. Addison-Wesley, Reading.

Desmedt, Y., Frankel, Y., 1989. Threshold cryptosystems. Crypto'89. *LNCS*, **435**:307-315.

Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T., 1999. Secure distributed key generation for discrete-log based cryptosystems. Eurocrypt'99. *LNCS*, **1592**:295-310.

Gruska, J., 1999. Quantum Computing. McGraw Hill, New York.

Herstein, I.N., 1964. Topics in Algebra. Blaisdell Publishing, Waltham, Massachusetts.

Ito, M., Saito, A., Nishizeki, T., 1987. Secret Sharing Scheme Realizing General Access Structure. Proc. IEEE Globecom'87, p.99-102.

Karnin, E.D., Greene, J.W., Hellman, M.E., 1983. On secret sharing systems. *IEEE Trans. Inf. Theory*, **29**(1):35-41. [doi:10.1109/TIT.1983.1056621]

Knuth, D.E., 1997. The Art of Computer Programming—Seminumerical Algorithms. Vol. 2, 3rd Ed., Addison-Wesley, Reading.

Koblitz, N., 1993. Introduction to Elliptic Curves and Modular Forms. Springer-Verlag, New York.

Kulesza, K., Kotulski, Z., 2002. On Secret Sharing Schemes with Extended Capabilities. RCMIS'02, **1**:79-88.

Kulesza, K., Kotulski, Z., Pieprzyk, J., 2002. On Alternative Approach for Verifiable Secret Sharing. Esorics'02. Available from IACR's Cryptology ePrint Archive (http://eprint.iacr.org/).

Kulesza, K., Kotulski, Z., 2003. On Automatic Secret Generation and Sharing for Karin-Greene-Hellman Scheme. *In*: Sołdek, J., Drobiazgiewicz, L. (Eds.), Artificial Intelligence and Security in Computing Systems Advanced Computer Systems. Kluwer Academic Publisher, Boston, p.281-292.

Li, C., Hwang, T., Lee, N., 1994. (*t*,*n*) threshold signature schemes based on discrete logarithm. Eurocrypt'94. *LNCS*, **950**:191-200.

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

Pedersen, T., 1991. A threshold cryptosystem without a trusted third party. Eurocrypt'99. *LNCS*, **547**:522-526.

Pieprzyk, J., Hardjono, T., Seberry, J., 2003. Fundamentals of Computer Security. Springer-Verlag, Berlin.

Shamir, A., 1979. How to share a secret. *Commun. ACM*, **22**(11):612-613. [doi:10.1145/359168.359176]

Shoup, V., Gennaro, R., 1998. Securing threshold cryptosystems against chosen ciphertext attack. Crypto'98. *LNCS*, **1403**:1-16.

Stadler, M., 1996. Publicly verifiable secret sharing. Eurocrypt'96. *LNCS*, **1070**:190-199.

APPENDIX A: PRELIMINARIES

KGH description

In KGH the secret is a vector of η numbers $\boldsymbol{S}_\eta=\{s_1, s_2, ..., s_\eta\}$. 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 $\boldsymbol{S}_\eta^{(j)}$ (j=1, 2, ..., t) with elements in $\mathbb{Z}_k$. To retrieve the secret they have to add the vectors component-wise in $\mathbb{Z}_k$.

For k=2, KGH method works like $\oplus$ (XOR) on η-bit 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.

In practice, it is often needed that only certain specified subsets of the participants should be able to recover the secret. The authorized set of participants is a subset of all participants. Participants from such set are able to recover the secret. The access structure describes all the authorized subsets. To design the access structure with required capabilities, the cumulative array construction can be used. Details can be found in (Ito *et al*., 1987; Pieprzyk *et al*., 2003). Combining cumulative arrays with KGH method, one obtains implementation of general secret sharing scheme (Pieprzyk *et al*., 2003).

Remarks about procedures and algorithms presented in the appendixes

Every routine is described in three parts:

(1) 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.

(2) 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 extension.

(3) Discussion (if needed). Methods and results are formally justified.

Preliminaries for Algorithms

1. Notation

As described in Section 2, random number generator and the Accumulator are needed. A secure communication channel and encapsulation have to be supported.

RAND yields $\boldsymbol{m}_i$ 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*($\boldsymbol{x}$) yields $ACC=ACC\oplus\boldsymbol{x}$ (performs bitwise XOR of *ACC* with the input binary vector $\boldsymbol{x}$, result is stored to *ACC*).

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

Basic Property: Let $\boldsymbol{m}_i$, i=1, 2, ..., n, such that

$$\bigoplus_{i=1}^{n}\boldsymbol{m}_i=\boldsymbol{0} \tag{A1}$$

form the set M. For any partition of M into two disjoined subsets C_1, C_2 ($C1\cup C2=M$, $C1\cap C2=\varnothing$), it holds:

$$\bigoplus_{\boldsymbol{m}_i\in C_1}\boldsymbol{m}_i=\bigoplus_{\boldsymbol{m}_i\in C_2}\boldsymbol{m}_i. \tag{A2}$$

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

2. Procedure description

GenerateM creates set of n binary vectors $\boldsymbol{m}_i$ satisfying Eq.(A1). Procedure is carried out by the Accumulator. The procedure returns $M=\{\boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_n\}$.

Procedure 1: *GenerateM*(n)

Accumulator:
ACC.reset;
for i=1 to n−1 do
$\boldsymbol{m}_i$:=*RAND*;
ACC.store($\boldsymbol{m}_i$);
save $\boldsymbol{m}_i$;
end // for
$\boldsymbol{m}_n$=*ACC.read*;
save $\boldsymbol{m}_n$;
return $M=\{\boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_n\}$;
end // *GenerateM*

Discussion We claim that the generated set M satisfies Eq.(A1). First, statistically independent random vectors $\boldsymbol{m}_i$ (i=1, 2, ..., n−1) are generated, while $\boldsymbol{m}_n = \bigoplus_{i=1}^{n-1} \boldsymbol{m}_i$, so

$$\bigoplus_{i=1}^{n} \boldsymbol{m}_i = \left(\bigoplus_{i=1}^{n-1} \boldsymbol{m}_i\right) \oplus \boldsymbol{m}_n = \left(\bigoplus_{i=1}^{n-1} \boldsymbol{m}_i\right) \oplus \left(\bigoplus_{i=1}^{n-1} \boldsymbol{m}_i\right) = \boldsymbol{0}.$$

Further in the paper whenever we make reference to set M, we mean the set as defined above.

APPENDIX B

This appendix contains procedures and algorithms for automatic generation and sharing of a random, prior nonexistent secret.

SetGenerateM description: It 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.

Authorized set replication (same cardinality sets)

The authorized set satisfies $|U^{(2)}|=|U^{(3)}|=n$, $U^{(2)} = \{\boldsymbol{s}_1^{(2)}, \boldsymbol{s}_2^{(2)}, ..., \boldsymbol{s}_n^{(2)}\}$, $U^{(3)} = \{\boldsymbol{s}_1^{(3)}, \boldsymbol{s}_2^{(3)}, ..., \boldsymbol{s}_n^{(3)}\}$. Algorithm *EqualSetReplicate* replicates set $U^{(2)}$ into the set $U^{(3)}$. It makes use of the procedure *SetReplicate*.

SetReplicate description: *SetReplicate* takes $U^{(2)}$ and M with cardinality $|M|=2|U^{(2)}|$. Hence $M=\{\boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_{2n}\}$. First, all participants $P_i^{(2)}$ are assigned corresponding vectors $\boldsymbol{m}_i$. Each of them performs bitwise XOR on their secret shares and corresponding $\boldsymbol{m}_i$. Operation result is sent to the Accumulator. Accumulator adds $\boldsymbol{m}_{i+n}$ to form $\boldsymbol{s}_i^{(3)}$, which is later sent to $P_i^{(3)}$. As the result, simultaneous creation and distribution of $U^{(3)}$ takes place.

Algorithm 1: *SetGenerateM*(d, n)

Accumulator:
GenerateM($d+n$)
for i=1 to d do // preparing $U^{(1)}$
$\boldsymbol{s}_i^{(1)}$:=$\boldsymbol{m}_i$;
save $\boldsymbol{s}_i^{(1)}$;
end // for
for i=d+1 to $d+n$ do // preparing $U^{(2)}$
j:=$i-d$;
$\boldsymbol{s}_j^{(2)} := \boldsymbol{m}_i$;
save $\boldsymbol{s}_j^{(2)}$;
end // for
return $U^{(1)} = \{\boldsymbol{s}_1^{(1)}, \boldsymbol{s}_2^{(1)}, ..., \boldsymbol{s}_d^{(1)}\}$,
$U^{(2)} = \{\boldsymbol{s}_1^{(2)}, \boldsymbol{s}_2^{(2)}, ..., \boldsymbol{s}_n^{(2)}\}$;
end // *SetGenerateM*

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

Accumulator:
n:=$|U^{(2)}|$;
for i=1 to n
send $\boldsymbol{m}_i$ to $P_i^{(2)}$;
$P_i^{(2)}$: $\boldsymbol{\omega}_i^{(2)} := \boldsymbol{s}_i^{(2)} \oplus \boldsymbol{m}_i$;
// $\boldsymbol{\omega}$ is an l-bit vector (local variable)
end // for
for i=1 to n
$P_i^{(2)}$ sends $\boldsymbol{\omega}_i^{(2)}$ to Accumulator;
Accumulator: $\boldsymbol{s}_i^{(3)} := \boldsymbol{\omega}_i^{(2)} \oplus \boldsymbol{m}_{i+n}$;
send $\boldsymbol{s}_i^{(3)}$ to $P_i^{(3)}$;
end // for
end // *SetReplicate*

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

EqualSetReplicate 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 in Section 4:

(1) $\bigoplus_{i=1}^{n} \boldsymbol{s}_i^{(3)} = \bigoplus_{i=1}^{n} (\boldsymbol{s}_i^{(2)} \oplus \boldsymbol{m}_i \oplus \boldsymbol{m}_{i+n})$

$= \left(\bigoplus_{i=1}^{n} \boldsymbol{s}_i^{(2)} \right) \oplus \left(\bigoplus_{i=1}^{2n} \boldsymbol{m}_i \right)_i = \bigoplus_{i=1}^{n} \boldsymbol{s}_i^{(2)}$ as requested.

(2) All $\boldsymbol{s}_i^{(3)}$ result from XOR of some elements from $U^{(2)}$ with random $\boldsymbol{m}_i$, $\boldsymbol{m}_{i+n}$, hence they are random numbers.

Authorized set replication (different cardinality sets)

For $|U^{(2)}| \neq |U^{(3)}|$ there are two possibilities:

Case 1 *SetReplicateToBigger* 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 $\boldsymbol{m}_i$ ($i>n$) not used by *SetReplicate*. As the result $U^{(3)}$, such $n=|U_2|<|U_3|=d$ is created and distributed.

Algorithm 3: *SetReplicateToBigger*($U^{(2)}$,d)

```
n:=|U^(2)|;
M:=GenereateM(d+n);
  Accumulator:
    SetReplicate(M, U^(2)) // assigns shares for participants
        // up to P_n^(3), it uses the first 2n elements of M
    for i=n+1 to d
        s_i^(3) := m_{i+n};
        send s_i^(3) to P_i^(3);
    end // for
end // SetReplicateToBigger
```

Discussion We claim that *SetReplicateToBigger* fulfils requirements stated in Section 4:

(1) First observe that:

$$\left[\bigoplus_{i=1}^{n} (\boldsymbol{m}_i \oplus \boldsymbol{m}_{i+n}) \right] \oplus \left(\bigoplus_{i=n+1}^{d} \boldsymbol{m}_{i+n} \right) = \left(\bigoplus_{i=1}^{2n} \boldsymbol{m}_i \right) \oplus \left(\bigoplus_{i=2n+1}^{d+n} \boldsymbol{m}_i \right) = \bigoplus_{i=1}^{d+n} \boldsymbol{m}_i,$$

so,

$$\bigoplus_{i=1}^{d} \boldsymbol{s}_i^{(3)} = \left[\bigoplus_{i=1}^{n} (\boldsymbol{s}_i^{(2)} \oplus \boldsymbol{m}_i \oplus \boldsymbol{m}_{i+n}) \right] \oplus \left(\bigoplus_{i=n+1}^{d} \boldsymbol{m}_{i+n} \right) = \left(\bigoplus_{i=1}^{n} \boldsymbol{s}_i^{(2)} \right) \oplus \left(\bigoplus_{i=1}^{d+n} \boldsymbol{m}_i \right) = \bigoplus_{i=1}^{n} \boldsymbol{s}_i^{(2)}.$$

(2) For $i>n$ all $\boldsymbol{s}_i^{(3)}$ are equal to random numbers $\boldsymbol{m}_i$. For $i \leq n$ all $\boldsymbol{s}_i^{(3)}$ result from XOR of some elements from $U^{(2)}$ with random $\boldsymbol{m}_i$, hence are random numbers.

Case 2 *SetReplicateToSmaller* 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 $\boldsymbol{s}_i^{(3)}$. First $d-1$ shares are sent to corresponding participants $P_i^{(3)}$. Remaining $\boldsymbol{s}_i^{(3)}$ ($i \in \{d, d+1, ..., n\}$) are combined to form $\boldsymbol{s}_d^{(3)}$ 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) : ω_i^(2) := s_i^(2) ⊕ m_i;
        // ω is an l-bit vector (local variable)
    end // for
    for i=1 to d-1
        P_i^(2) sends ω_i^(2) to Accumulator;
        Accumulator: s_i^(3) := ω_i^(2) ⊕ m_{i+n};
        send s_i^(3) to P_i^(3);
    end // for
  ACC.reset
    for i=d to n  // all ω_i^(3) for i≤d were already used
        P_i^(2) sends ω_i^(2) to Accumulator;
        Accumulator: ACC.store(ω_i^(2))
    end // for
  s_d^(3)=ACCA.read;  // s_d^(3) := ⊕_{i=l}^{n} ω_i^(2)
  send s_d^(3) to P_d^(3);
end // SetReplicateToSmaller
```

Discussion We claim that *SetReplicateToSmaller* fulfils requirements stated in Section 4:

(1) First observe that:

$$\left[\bigoplus_{i=1}^{d-1} (\boldsymbol{m}_i \oplus \boldsymbol{m}_{i+n}) \right] \oplus \left(\bigoplus_{i=d}^{n} \boldsymbol{m}_i \right) = \left(\bigoplus_{i=1}^{n} \boldsymbol{m}_i \right) \oplus \left(\bigoplus_{i=1}^{d-1} \boldsymbol{m}_{i+n} \right) = \left(\bigoplus_{i=1}^{n} \boldsymbol{m}_i \right) \oplus \left(\bigoplus_{i=n}^{n+d-1} \boldsymbol{m}_i \right) = \bigoplus_{i=1}^{n+d-1} \boldsymbol{m}_i,$$

so,

$$\bigoplus_{i=1}^{d} \boldsymbol{s}_i^{(3)} = \left[\bigoplus_{i=1}^{d-1} (\boldsymbol{s}_i^{(2)} \oplus \boldsymbol{m}_i \oplus \boldsymbol{m}_{i+n}) \right] \oplus \left[\bigoplus_{i=d}^{n} (\boldsymbol{s}_i^{(2)} \oplus \boldsymbol{m}_i) \right] = \left(\bigoplus_{i=1}^{n} \boldsymbol{s}_i^{(2)} \right) \oplus \left(\bigoplus_{i=1}^{n+d-1} \boldsymbol{m}_i \right) = \bigoplus_{i=1}^{n} \boldsymbol{s}_i^{(2)}.$$

(2) All $\boldsymbol{s}_i^{(3)}$ result from XOR of some elements from $U^{(2)}$ with random $\boldsymbol{m}_i$, hence they are random numbers.