Improvements to Distributed Key Generation For Use in a Real-World Setting

An EPFL IN Semester Project

Cedric Cook

Supervisor: Nicolas Gailly

Professor: Bryan Ford

12 January 2017

Contents

1 Introduction

In cryptography, secret sharing is a wide spread concept that is needed as a foundation for a multitude of distributed cryptographic systems. For example in security critical applications, just one password may not be enough to secure access to a resource, and hence the concept of distributed keys is created to spread risk over multiple agents in a system. In order to distribute the individual parts of such a master key and to use this key in a secure and intrusion-proof manner, a protocol called Distributed Key Generation (`DKG`) is used. The DKG protocol can be implemented programmatically by using multiple instances of Verifiable Secret Sharing (`VSS`). In this project we will take an existing implementation of DKG that was realized in the Go programming language as a part of the Kyber Advanced Crypto Library, and implement changes to make the implementation more compatible with a real-world internet-like asynchronous setting. In the following sections we will first go more in depth on the topics of VSS and DKG, and then address the adjustments to make to the scheme before going over the actual implementation in Go and the corresponding tests. Finally we discuss the outcomes and possible future work.

2 A quick introduction to VSS and DKG

Distributed Key Generation allows a set of n agents (or servers) to collectively compute a public-private keypair without any single agent having access in terms of computation and storage to the private key, and without necessity of trusting a third party to handle the process. After such a scheme has been completed, the public key is publicly accessible and usable, whilst the attacker has to comprise a larger number of clients in order to recover the key. In this paper we are specifically interested into (n, t)-threshold variants of DKG and VSS, implying that for n agents participating in the protocol, an attacker needs to compromise at least t of the agents in order to know or use the private key. The resulting public key of this scheme may be used by any party, even external, to encrypt a secret that may only be decrypted with participation of at least t of the agents, whereas the distributed private key can be used for collectively signing documents or software without this key being reconstituted in a single location.

DKG is a cryptographic scheme that makes use of other cryptographic schemes. The most basic of these is secret sharing, a method with which a secret can be distributed amongst agents, each of whom is allocated a share. In secret sharing, agents are either the dealer, or a participant. The dealer has complete knowledge, i.e. the dealer has both the public key and the private key. A commonly known threshold secret sharing scheme was invented by *Shamir* [Sha79] and is information-theoretically secure and thus remains secure even if the adversary has unlimited computing power. The scheme is based principally on Lagrange interpolation, which lets us find the unique degree $(t-1)$ polynomial given exactly t points. By Shamir, if a $t-$threshold secret sharing system is needed, a polynomial $f()$ of $(t-1)$ degree is created, where the first coefficient is equal to the complete secret (private key), and all coefficients of remaining degree are picked randomly. Then, for each participant i the point on the polynomial $f(i)$ is calculated, and the result is communicated via a private secure channel to the participant. If t participants reveal their respective points, the polynomial $f()$ can thus be found and calculating $f(0)$ will return the private secret.

The first issue that arises with secret sharing is that the dealer is assumed to be honest which preferably we do not want to assume. Furthermore, the participants may lie about their shares to gain information about the other shares. Verifiable Secret Sharing is a scheme that lets participants verify the consistency of each others' shares, and ensures the protocol continues to work even if the dealer is malicious. VSS first works similarly to secret sharing, in that the same polynomial $f()$ is created, and the shares are again transmitted securely to each participant. Then however, the dealer also publishes a list of commitments of the coefficients of the polynomial. These commitments are created as follows: in the beginning of the protocol a generator g of a mathematical cyclic group G is chosen, where computing discrete logarithms in G is computationally hard. Each commitment is calculated as g raised to the power of a coefficient of $f()$.

Since calculating discrete logarithms is hard in G, it is "impossible" to compute $log(g^a)$ where a is any coefficient of $f()$. However, to verify that the value v_i revealed by a participant i is a valid share, any agent can check that $g^{v_i} = \prod_{j=0}^{t} c_j^{i^j}$. If this is not the case, then the share is faulty. The agent holding the faulty share will broadcast a complaint against the dealer. The dealer then has a chance to show its correctness, by broadcasting a justification which reveals the private share v of the complaining agent. Agents can then once again check against the commitments, and prove that either the dealer is malicious and disqualified, or the share is correct. If at least t agents have verified each others' shares, than they can find the corresponding polynomial, and thus compute the secret.

2.1 DKG protocols

In VSS a single agent is the dealer and other agents are the participants. If the goal of the protocol is to generate a private key that is not known to any single party, then a problem arises in that the VSS dealer *does* know the secret. To solve this problem DKG essentially runs n copies of VSS in parallel where in every copy another agent is the dealer. Each dealer picks a random secret to share, and the overall secret is the sum of all random secrets. The public value or public key is the product of each individual public key. Finally each agent knows their part of the secret, and a share of every other secret, but no one agent has the whole combined secret. DKG works on a round basis, and in Pedersen's implementation, one round is needed in absence of faults to establish the shared public-private keypair.

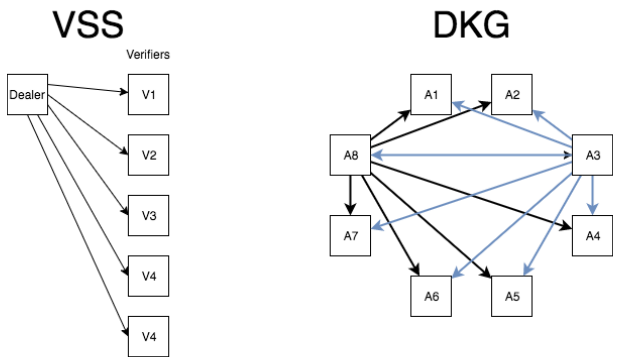

Figure 1: VSS represented by the dealer and the remaining agents (verifiers). In the DKG diagram only the shares distributed by agent A8 and A3 are shown, but all agents function similarly.

2.2 DKG preliminaries

Communication Model As stated before, all pairs of agents in the system have private and authenticated channels between them. In addition, the whole system shares a broadcast channel to which all agents have access. In the current model, computation on messages happens in synchronized rounds, where all agents share synchronized clocks and messages sent in round a round i are received before the beginning of round $i+1$. It must be noted that exactly this assumption will be discussed and adjusted in this project.

Adversary The adversary is assumed to be able to corrupt a maximum of $(t-1)$ of the n agents in the system. A minimum value for t is taken to be $t < \frac{n}{2}$ for guaranteed secrecy and robustness of the system. The adversary can read corrupted parties, and make them apply any arbitrary protocol instead of, or in complement to the agreed DKG protocol.

3 Problem statement

The limitations of the current implementation of the DKG protocol in the kyber package are such that the protocol is not performant in a real-world setting. Therefore we will study the synchronized network setting, and adjust the protocol in order to address these limitations. Our proposal is to add a signal to the protocol that indicates round termination, after which we also add methods that define the behaviour after such a termination signal. This behaviour ensures that at the end of each round, each agent has guaranteed responses for each other agent, and so the protocol can continue execution.

In the following sections we will first introduce the kyber package properly, as well as its current implementation of a DKG protocol, explicit the limitations of the setting, and the adjustments that were made to overcome these limitations.

3.1 Kyber

Kyber is an "Advanced Crypto library in the Go language" under ongoing development by the DEDIS lab at EPFL [1]. The library provides cryptographic primitives and higher-level building blocks for cryptography applications that surpass signing and encryption. It defines an interface for mathematical groups and cryptographic suites, and those in turn are used for a host of packages such as signatures, cryptographic proofs, verifiable cryptographic shuffles, and the share package, topic of this project, that provides verifiable threshold cryptographic schemes.

As part of the share package two sub-packages are of interest to us: `vss` and `dkg`. Both of these packages have an implementation that follows Pedersen's method and another that follows Rabin's method [Rab07]. In the following parts of this report particularly Pedersen's implementation will be discussed, but the work was extended to Rabin's implementation and does not differ notably.

3.2 Limitations of the synchronized network setting

We recall from section 2.1 that Pedersen's DKG protocol uses n parallel Feldman-VSS instances and combines the secrets of each dealer to create the overall secret. Together, the agents that shared their secret correctly form the set $QUAL$. To formalise the protocol, we explicit it in Figure 2.

As we can observe, one run of VSS is necessary per instance to successfully create the distributed keys. Each run of VSS is defined to take one round of communications to broadcast the commitments and individually send the shares to all participants, as well as treat any complaints the participants may have and the dealer's corresponding justifications. The major assumption that is made here was defined in the DKG preliminaires earlier (section 2.2), and that is that the communication channels are synchronous, and thus all participants

[1] https://github.com/dedis/kyber

Protocol JF-DKG

1. Each agent A_i (as a dealer) chooses a random polynomial $f_i(z)$ over Z_q of degree t:
$$f_i(z) = c_{i0} + c_{i1}z + \cdots + c_{it}z^t$$
A_i broadcasts $C_{ik} = g^{c_{ik}} \bmod p$ for $k = 0, \dots, t$. Denote c_{i0} by z_i and C_{i0} by y_i. Each A_i computes the shares $s_{ij} = f_i(j) \bmod q$ for $j = 1, \dots, n$ and sends s_{ij} secretly to agent A_j.

2. Each A_j verifies the shares he received from the other parties by checking for $i = 1, \dots, n$:
$$g^{s_{ij}} = \prod_{k=0}^{t} (C_{ik})^{j^k} \mod p \tag{1}$$
If the check fails for an index i, A_j broadcasts a *complaint* against P_i.

3. Agent A_i (as the dealer) reveals the share s_{ij} matching (1) for each complaining agent A_j. If any of the revealed shares fails this equation, A_i is disqualified. We define the set $QUAL$ to be the set of non-disqualified agents.

4. The public value y is computed as $y = \prod_{i \in QUAL} y_i \mod p$. The public verification values are computed as $C_k = \prod_{i \in QUAL} C_{ik} \mod p$ for $k = 1, \dots, t$. Each agent A_j sets his share of the secret as $x_j = \sum_{i \in QUAL} s_{ij} \mod q$. The secret shared value x itself is not computed by any party, but it is equal to $x = \sum_{i \in QUAL} z_i \mod q$.

Figure 2: Pedersen's DKG protocol.

are guaranteed to receive the share and commitments from the dealer, and the dealer is guaranteed in turn to receive the complaints. Additionally, in the kyber implementation, the participants either broadcast a complaint about the dealer, or they broadcast a positive response to acknowledge reception of their share and the commitments (called a *Deal*), and approve the correctness thereof.

It is clear that in the real world this assumption of complete synchronicity is relatively strong. If we were to consider DKG, or VSS, in a real world setting we might think of each agent in the scheme as being a distinct server, in any geographical location in the world, in any type of logical system. Hence, a pair of agents can be separated not only by a physically large distance, they may also be separated in terms of networking and communications by a scala of connection devices such as bridges, routers and other intermediaries. This, combined with the overall state of congestion on the internet and the underlying protocols, leads us to a situation in which it is very well possible for communications to be asynchronous, and possibly for messages to not even arrive.

If we consider these restrictions it follows that we should try to relax the synchronous communication assumption. We thus design an adjustment for VSS and DKG that allows for the round to finish whether all parties respond or not, after a finite time.

3.3 Situation

Before In the current version of the VSS implementation, the dealer sends a share individually to each participant, and broadcasts the commitments. Each participant then processes the *Deal* (i.e. their share and public commitments), verifying their share against the public commitments. If the verification is successful, they will respond by broadcasting a response specific to this participant containing *StatusApproval* as shown in Figure 3. If the verification was not successful the participant's response will contain *StatusComplaint*. Then, the dealer takes all responses, and for any response from a participant i that has *StatusComplaint* it will reveal the corresponding share s_i in form of a *Justification*. The justification is broadcast, and all participants can check that it is correct, as shown in Figure 4. Therefore, at the end of one round of this VSS, the dealer has received responses that are either correct, or they have justified themselves against the participant. The participants meanwhile will have received their correct share, or have complained, then received the correct share in justification or marked the dealer as malicious.

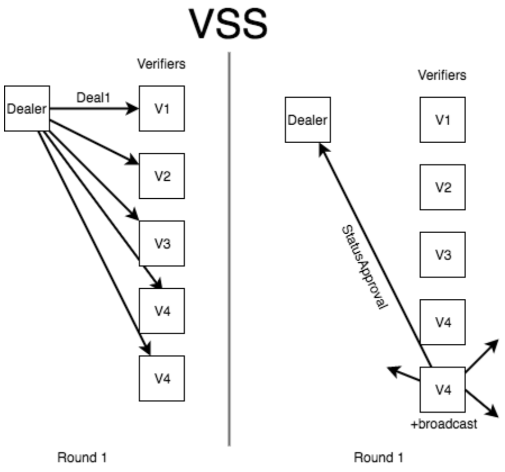

Figure 3: A partial VSS round where a verifier approves the Deal.

After As we saw in the previous paragraph, the round blocks until the dealer has received responses with approvals from all verifiers, or at least until t verifiers have responded positively. Since transmitting both the *Deal* from the dealer to the participant, and the response in the opposite direction may suffer from communication problems, this round can block for a long time. Our

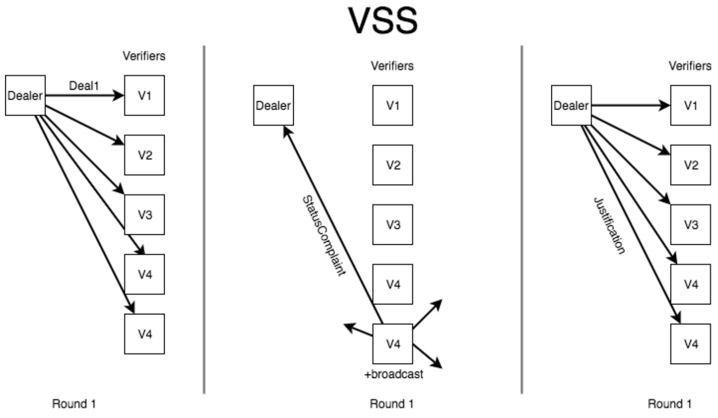

Figure 4: A partial VSS round where a verifier complains, and the dealer justifies the verifiers' Deal.

proposal is to add a signal to both the dealer and the participants to indicate that a maximum time has been reached, and thus the round must be completed. All participants that have not responded with a *StatusApproval* after this signal are considered to be no longer participating to the scheme for this execution, either by communications faults or by malicious takeover of an adversary. The adversary may for example have changed the participant's protocol to simply not acknowledge reception of a *Deal*. The participants that have not responded after the signal are said to be timed-out. Since the participation of these timed-out parties is equally unimportant as malicious parties we treat them similarly.

3.4 Implementation changes to VSS

To understand the changes that are applied to the VSS implementation in kyber, we present in Figure 5 the kyber representations of the *Dealer* and *Verifier* structures, and the *Aggregator* structure they both use.

The exact changes we apply to the VSS implementation are the following: We add a method *SetTimeout* to both the *Dealer* and *Verifier* structures. Both the Dealer and Verifier objects use an underlying *aggregator* to manage responses of other participants, and therefore the SetTimeout method simply calls a further method namely *cleanVerifiers* on the aggregator object. The SetTimeout method models the signal that we evoked in the previous paragraph, which indicates that the round has now reached its maximum running time. The aggregator holds a list of responses of all participants and the cleanVerifiers method sets a response with *StatusComplaint* for all verifiers for which we do not have a response with *StatusApproved* yet. Thus, after a protocol that utilises VSS has called the SetTimeout method, the respective agent has a guaranteed response for each participant, whether that response is positive or negative.

The last change to make to VSS to complete our relaxation of the syn-

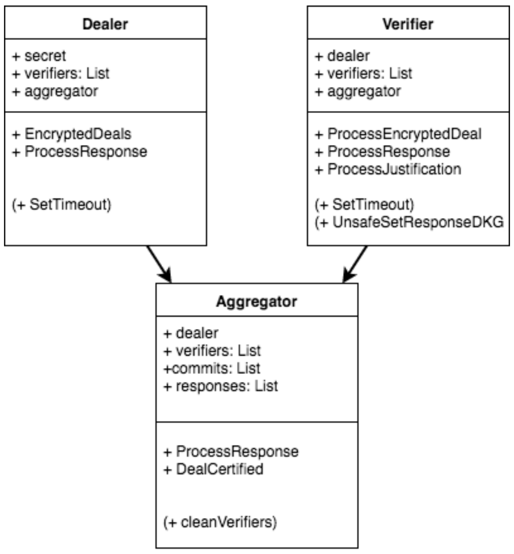

Figure 5: An abbreviated schema of the *Dealer*, *Verifier* and *Aggregator* structures and their fields and methods in the kyber package. The methods indicated between parentheses are part of this projects' modifications and described in the text.

chronous assumption is in the existing *DealCertified* method. Before, as the protocol demands, this method returns the state of validity of the sharing scheme, by checking that the number of complaints is less the threshold, and that the dealer has not been found malicious by the current agent. Additionally, this method also only returns positively if there are at least $\geq t$ responses with *StatusApproval*. Since we have altered the meaning of the *StatusComplaint* slightly, we now let the method return true if no response is missing, at least $\geq t$ responses are positive, and the dealer is not indicated malicious. Now that we have completed the changes to VSS, we can address the adjustments to make to DKG.

3.5 Implementation changes to DKG

In the DKG implementation a single structure is used to represent one agent; the *DistKeyGenerator*. This structure consists of one VSS Dealer, as well as n VSS Verifiers. The dealer here is the agent itself, and the verifiers are all agents, who each have a DistKeyGenerator structure in which they are the dealer themselves and the n verifiers, and so on. Thus we can consider the DistKeyGenerator structure to be the basis for one of the n parallel copies of VSS that will be executed. Since most of the heavy lifting in the DKG protocol is done by VSS, there are only few changes to make to it. We add a *SetTimeout* method to DKG like we did to VSS before. The method is added to the DistKeyGenerator structure and it allows the user of the protocol to indicate the end of the round, after which the method will call *SetTimeout* on all the verifiers that are contained in the DistKeyGenerator, which we implemented in section 3.4.

After an agent a_i has distributed deals to all other agents, it sets a response for the verifier that represents itself, but does not broadcast this response to all other agents. This produces a problem in that another agent a_k in the scheme will receive a deal from a_i, and treat it in some way, but never set the response for a_k's verifier that represents a_i, since that response was never sent by a_i. Since our new implementation would set a *StatusComplaint* for a_i as it has not sent a response, we would falsely disqualify a_i. Therefore, we implement the following change: whenever an agent a_k receives a deal from agent a_i, a_k will directly set a response with *StatusApproval* for the verifier representing a_i. Allowing this behaviour can possibly compromise the protocol, so we call this method a unsafe method; named *UnsafeSetResponseDKG*. It is implemented on VSS Verifier, and allows another application or scheme to manually set a response instead of automatically setting it after a response was received.

4 Testing

In the kyber go package code integrity is preserved by means of testing amongst others. At the time of the start of this project the overall code coverage percentage was between 80 - 87% for the `vss` and `dkg` packages, and thus most of the statements are covered. Since we added logic to the code base, tests had to be added to verify the new statements. Tests were created for all new code, and a such the overall coverage has increased.

First off, it was of importance to check what the existing tests were checking for, and if, after the applied changes, they no longer functioned, to modified them to reflect the changes. In VSS three existing test cases were found that needed to now use the new *SetTimeout* method. The principal reason for this is that previously it was possible to check if the scheme's secret could be recovered or not before finishing a round. Since we have left the synchronous setting the protocol requires that a response has been set for each verifier, and thus where before it simply ignored all unresponsive verifiers, we now need to explicitly tell the protocol that the round is finished and to execute the necessary actions therefore.

Then we needed to create test that would correctly check the functionality of the *SetTimeout* method and its dependants, as well as the changed functionality in *DealCertified*. In order to do this, a test was created that checks the functionality on the aggregator of the dealer. The test sets manual responses for t threshold number of verifiers, and does nothing for the remaining $n - t$ verifiers. Theoretically now enough responses are present and this is checked by the test. But since no information is available about the remaining verifiers, the Deal should not be certified as that requires responses for all verifiers. Then, the *SetTimeout* method is called on the dealer, and it is verified that now the Deal is indeed certified. In a very similar way the functionality of the *SetTimeout* method on the verifier is tested, and thus the description is left out of this report but available in the source code of the kyber package.

Since we also modified the DKG code, it is necessary to test that new method as well. The test for this purpose implements and almost complete DKG exchange with a small exception. Deals are created by each agent and processed by all other agents, which generates a response for each agent. Instead of now distributing these responses to all other agents which would simulate the standard broadcast, the test distributes each response to another agent only if that agent does not have enough approvals for the related Deal yet, and else it ignores the agent. Therefore, each agent has exactly the threshold number of approvals for each Deal, but no responses from the remaining $n - t$ verifiers for this Deal. This is verified by checking if for each agent a_i it holds true that none of the other agents are in its $QUAL$ set. Any other agent a_k will not be in the set when they have not certified the deal. As some verifiers have not responded, the deal is not certified, and thus the agent is not in the $QUAL$ set as expected. Afterwards the *SetTimeout* method is called on each agent, and now it is checked that all agents have every other agent in the *QUAL*, which should be the case.

5 Discussion

The modifications that were made to the kyber package have been implemented as well as the corresponding tests. The project proposed a variety of challenges, of which the principal was the complexity of the cryptographic schemes. The VSS and DKG protocols that are used in this project are non-trivial to continuously conceptualise due to, amongst other reasons, the fact that each agent holds a complete representation of the rest of the agents in the scheme and the messages that were exchanged, and therefore it makes it difficult to reason on the problem and the solution, even when the overall limitations seem very clear and the solution so evident. The limited cryptographic knowledge of the author meant that a larger amount of time was needed for comprehension of the cryptographic bases of the schemes.

We point out that the changes made to the DKG protocol's implementation have no great negative impact on the performance of the scheme. The logic to verify if the key generation is valid is only slightly more complex, but because of the relaxation of the synchronicity assumption, there is an augmented chance of this protocol working well in a more "in the wild" setting. The changes to the protocol have adjusted the meaning of a negative status in the response to a Deal, but this has no impact on the use of the response to complain about a given Deal. As before the changes, the complaint and justification mechanism continues to work, but after the round has ended a negative response has a double meaning, i.e. either the verifier did not accept the Deal, or the verifier never responded, and in both cases the verifier is disqualified from the exchange, therefore the scheme continues to work as expected.

As the target of this project was to improve the DKG scheme in the kyber package, some further work is proposed in the form of a Share Renewal mechanism. In the current version of the protocol the shares are distributed one time only, and then permanently remain static for the related public key. If we consider that such a public key may be used in the long term to protect some data, then it may be useful to be able to create new shares, that without changing the longterm public key, ensure that an agent has to continuously be non-malicious in order to keep participating in the scheme. Such an implementation of share renewal may follow [Her95].

6 Conclusion

The target of this project was to asses weak points in the kyber implementation of DKG, and to find a solution to improve on these points. In particular, the problem of the synchronous assumption that exists in the original protocol was addressed. This assumption makes it difficult to use this protocol in a real-world setting, and thus this project intended to remove or reduce this to a weaker assumption. The solution that was proposed and implemented was to add a signal to the protocol's agents to indicate a timeout, and to handle the rest of the protocol accordingly. Tests were also added to allow for continuous checking of correct functioning of the added functionality.

To conclude, this project has managed to address at least one weakness for making DKG usable on the internet successfully, and may serve as a basis for further improvements.

7 References

References

[Sha79] Adi Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613. DOI: `https://dl.acm.org/citation.cfm?doid=359168.359176`.

[Her95] Amir Herzberg. "Proactive Secret Sharing". In: *Crypto '95* LNCS.963 (1995), pp. 339–352.

[Rab07] Tal Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *Journal of Cryptology* 20 (Oct. 2007), pp. 51–83.