Practical Large-scale Distributed Key Generation

John Canny and Stephen Sorkin

University of California, Berkeley, CA 94720 USA
{jfc,ssorkin}@cs.berkeley.edu

Abstract Generating a distributed key, where a constant fraction of the players can reconstruct the key, is an essential component of many large-scale distributed computing tasks such as fully peer-to-peer computation and voting schemes. Previous solutions relied on a dedicated broadcast channel and had at least quadratic cost per player to handle a constant fraction of adversaries, which is not practical for extremely large sets of participants. We present a new distributed key generation algorithm, sparse matrix DKG, for discrete-log based cryptosystems that requires only polylogarithmic communication and computation per player and no global broadcast. This algorithm has nearly the same optimal threshold as previous ones, allowing up to a $\frac{1}{2} - \epsilon$ fraction of adversaries, but is probabilistic and has an arbitrarily small failure probability. In addition, this algorithm admits a rigorous proof of security. We also introduce the notion of matrix evaluated DKG, which encompasses both the new sparse matrix algorithm and the familiar polynomial based ones.

Keywords: Threshold Cryptography. Distributed Key Generation. Discrete Logarithm. Random Walk. Linear Algebra.

1 Introduction

Distributed key generation (DKG) is an essential component of fully-distributed threshold cryptosystems. In many contexts, it is impractical or impossible to assume that a trusted third party is present to generate and distribute key shares to users in the system. In essence, DKG allows a set of players to collectively generate a public/private key pair with the "shares" of the private key spread over the players so that any sufficiently large subset can reveal or use the key. The generated key pair is then used in a discrete-log based threshold cryptosystem. Commonly the security parameter of such a system is called the threshold, t. This is the number of players that can be corrupted without the key being compromised.

Most distributed key generation schemes in the literature do not carefully consider the communication and computation cost required of each server. Specifically, most schemes require $O(nt)$ computation and communication per player, where n is the number of players participating in the scheme. In this paper, we present a randomized algorithm called sparse matrix DKG that reduces this cost to $O(\log^3 n)$ per player, both in terms of computation and communication, in the presence of $\Omega(n)$ malicious parties. For large systems, this difference is quite significant. The cost of this gain is a slight chance that the algorithm fails. We formalize this cost in the definition of a probabilistic threshold distributed key generation algorithm. We also show how sparse matrix DKG is a specific instance of a more broad family of DKG algorithms.

2 Basic System Model

The systems we describe involve a group of n players. The players are modeled by probabilistic polynomial-time Turing machines. They are connected with secure point-to-point channels, but without any broadcast channel. We feel that these assumptions are realistic for practical situations: true broadcast is only achievable in practice using Byzantine agreement but secure channels can be achieved through a public key infrastructure. We also assume that the players have access to a common source of randomness. The adversaries in this system are static and are assumed to be fixed when the algorithm begins. This is a reasonable assumption since, in practice, this algorithm is sufficiently fast that successful dynamic adversaries are unlikely.

For simplicity, we assume that there is an honest, but not trusted, dealer present to aid in the initialization of the algorithm. The first task of the dealer is to establish the set of n players that will participate in the algorithm. This task makes the dealer resemble a directory server for the players. The users who wish to generate a distributed key are assumed to know the identity of the dealer. These users will then contact the dealer in order to secure a unique identifier for the algorithm. This dealer may also assist in the creation of a public key infrastructure so that users can transmit messages securely through encryption or authenticate messages with digital signatures. The dealer then decides when enough users are present for the algorithm to begin. This is decided by either having a fixed number of users known up front or requiring that users who wish to participate send a message to the dealer within a fixed time limit. In either case, this process determines the value of n, the number of users participating in the algorithm. Part of our initial assumption is that the dealer has access to a random oracle [1] so some randomness may be distributed to the players. Based on the value of n, the dealer will decide on a set of random bits and distribute a subset of them to each player.

In practice, the dealer need not be a single party, and could be implemented through a logarithmic size group of users or a hierarchy of users.

3 Distributed Key Generation Protocols

3.1 Previous Work

Existing literature on DKG algorithms is quite broad. One main line of approach began with the polynomial-based algorithm presented by Pedersen [8]. This algorithm was presented by other authors in varied forms as the basis of various threshold cryptosystems. This basic algorithm and these modifications, however, were vulnerable to a variety of attacks that would allow an adversary to bias the distribution of the private and public keys. This flaw was remedied by Gennaro, et al. [4] in a protocol that operates in two phases and uses Pedersen's verifiable secret sharing algorithm [9] to protect the bit commitments of the public key against static adversaries. All of these approaches require $O(t)$ broadcasts and $O(n)$ point-to-point messages for each player. Gennaro, et al. followed up their paper [5] with an explanation of how Pedersen's original algorithm is secure when used for Schnorr signatures. One main advantage for using the basic Pedersen algorithm is saving one broadcast round.

3.2 Definitions

The following definitions apply to discrete-log based cryptosystems. The globally known constants are p, a large prime; q, a large prime that divides $p-1$ and g an element of order q in Z_p. The first three criteria of the following definition have been used widely to define DKG protocols. The fourth was added by Gennaro, et al. in order to quantify the secrecy of an algorithm's key against malicious participants in the generation phase.

Definition 1. *A t-secure distributed key generation algorithm satisfies the following requirements, assuming that fewer than t players are controlled by the adversary:*

(C1) All subsets of $t+1$ shares provided by honest players define the same unique secret key x.
(C2) All honest parties have the same value of the public key $y = g^x \mod p$, where x is the unique secret guaranteed by (C1).
(C3) x is uniformly distributed in Z_q (and hence y is uniformly distributed in the subgroup generated by g).

(S1) The adversary can learn no information about x except for what is implied by the value $y = g^x \mod p$.

We propose the following modification to allow a DKG algorithm to fail with small probability. This will allow for algorithms that are considerably more efficient with arbitrarily small impacts to security.

Definition 2. *A probabilistic threshold (α, β, δ) distributed key generation algorithm satisfies requirements (C2), (C3) and (S1) above as well as (C1'), (C4') below (which replace (C1)), all with probability $1-\delta$, assuming that the set of players controlled by the adversary is less than αn.*

(C1') The shares of any subset of honest players define the same key, x, or no key at all.

(C4') Any subset of at least βn honest players can recover the key with probability $1-\delta$.

By these definitions, a t-secure DKG algorithm is a probabilistic threshold $\left(\frac{t}{n}, \frac{t+1}{n}, 0\right)$ DKG algorithm as well.

3.3 Matrix Evaluated DKG

The sparse matrix distributed key generation algorithm we propose for very large sets of players is a specific instance in a family of protocols which we call matrix evaluated DKG. In section 3.4 we show that this family includes the familiar DKG algorithm introduced by Gennaro, et al. This general technique consists of three primary phases. In the first phase, all the players create their secrets and share them with the others. After it is decided which players have correctly shared their secrets, the public key is recovered. Finally, after the generation is complete, the algorithm provides a method for recovering the secret with only a subset of the shareholders. The use of a matrix to codify the relation between the key and its shares is similar to the technique Blakley proposed for secret sharing [2].

Master Algorithm

1. Start with a dealing phase so that all players know E, v, g, h, p, q where p is a large prime, q is a large prime that divides $p-1$, g and h are elements of order q in Z_p, E is an $m \times n$ matrix over Z_q and v is an m element vector over Z_q.
2. **Generate** x:
   (a) Each player i chooses two row vectors, $a_i, a'_i \in Z_q^m$.
   (b) Player i then calculates $s_i \triangleq a_i E, s'_i \triangleq a'_i E$. Define the *checking group*: $Q_i = \{j | s_{ij} \neq 0 \vee s'_{ij} \neq 0\}$. Player i sends the jth element of each, s_{ij}, s'_{ij} to player $j \in Q_i$ and broadcasts[1] $C_i \triangleq g^{a_i} h^{a'_i} \mod p$ (where both the exponentiation and multiplication are element-wise) to all $j \in Q_i$.
   (c) Each player j verifies the shares received from other players. For each i such that $j \in Q_i$, player j checks if:
   $$g^{s_{ij}} h^{s'_{ij}} = \prod_{k=1}^{m} (C_{ik})^{E_{kj}} \mod p \tag{1}$$
   If this check fails for i, player j broadcasts a complaint against player i to Q_i.
   (d) Every player i who was complained against by player j will broadcast s_{ij}, s'_{ij} to Q_i.
   (e) The other players in Q_i will check the broadcast s_{ij}, s'_{ij} and mark as invalid each i for which Eq. 1 does not hold.
3. Each player i builds the set V of all players who were not marked invalid and sets their share of the secret as $x_i \triangleq \sum_{j \in V} s_{ji}$. Note that x_i is the ith element of the vector $\left(\sum_{j \in V} a_j\right) E$. The secret key is defined as $x \triangleq \left(\sum_{i \in V} a_i\right) v$.
4. **Reveal** $y = g^x \mod p$
   (a) Each valid player $i \in V$ broadcasts the vector $A_i \triangleq g^{a_i} \mod p$ to Q_i.

[1] All the following broadcasts are to Q_i only. In the sparse matrix algorithm, $|Q_i| = O(\log n)$.

(b) Each player j verifies that each A_i is valid by checking for each $i \in V$ such that $j \in Q_i$ if:

$$g^{s_{ij}} = \prod_{k=0}^{m} (A_{ik})^{E_{kj}} \mod p \tag{2}$$

If this check fails for i, player j broadcasts a complaint against player i as well as s_{ij}, s'_{ij} to Q_i.

(c) If at least one valid complaint is filed against player i (Eq. 1 holds for s_{ij}, s'_{ij} but Eq. 2 does not), then all players in Q_i will reconstruct a_i in public by solving a set of linear equations ($s_i = a_i E$), for the valid values of s_i.

(d) Each of the honest members of Q_i knows both whether player i is valid and, if so, the correct value A_i. To find y, it suffices to ask each member of each Q_i, find the set V, and then take $y \triangleq \prod_{i \in V} \prod_j A_{ij}^{v_j}$.

Sharing the secret The basis of this algorithm is what we call an *evaluation matrix* E, which has m rows and n columns, where n is the total number of players in the system. Each player i picks an internal secret a_i, which is a row vector with m elements and, from that, creates an external secret $s_i \triangleq a_i E \mod q$, another row vector, now with n elements. Player i then reveals the jth column of the external secret s_{ij} to player j. If the a_i and E are structured, it is possible that fewer than n users are assigned non-zero shares by player i.

In order to demonstrate that player i is creating consistent shares of the external secret, she must broadcast a committed version of her internal secret. For this we employ the VSS scheme introduced by Pedersen [9]. Player i creates another, random internal secret a'_i and broadcasts a commitment vector $C_i = g^{a_i} h^{a'_i} \mod p$ (the multiplication here is element-wise) to the *checking group* Q_i of players who receive the non-trivial shares. Each member of Q_i can then verify that her share of s_i was created from the same a_i as the others. This is achieved by each player j checking that Eq. 1 holds for each i, since all honest players are assumed to agree on C_i

This equality will certainly hold if $s_{ij} = \sum_{k=1}^{m} a_{ik} E_{kj}$ as specified above. If it does not, then the secret share is invalid and player j broadcasts a complaint against player i. In response, player i will broadcast s_{ij}, s'_{ij} to demonstrate that Eq. 1 is satisfied. If player i broadcasts a pair of secret shares that did not satisfy Eq. 1, then the other players will mark i as invalid, since the external secret of player i is not consistent with any internal secret, and hence it will be impossible to recover the secret key if it includes this share.

At this point, V is well defined and each checking group Q_i knows whether $i \in V$, since the decision is based on broadcast information. At this point both the global private key x and public key y are well defined by the following equations, where T is a linear function (e.g., an inner product with a fixed vector, v):

$$
\begin{aligned}
a &\triangleq \sum_{i \in V} a_i \\
x &\triangleq \sum_{i \in V} T(a_i) \\
&= T(a) \\
y &\triangleq g^x \mod p
\end{aligned}
$$

Revealing the public key Note that the public key is not yet publicly known. This is necessary to ensure that an attacker does not skew its distribution by manipulating the set V. After V is established, the public key may be safely extracted. Every player $i \in V$ broadcasts their share of the public key, the vector $A_i = g^{a_i} \mod p$. The other players will check if Eq. 2 holds.

Like the previous check, if the rules of the algorithm are followed, the equality holds. If it does not and player j broadcasts a complaint, the allegation of cheating is treated with more caution than before. Specifically, player j must prove that her complaint is valid by broadcasting s_{ij}, s'_{ij}. The other players will check that Eq. 1 holds for these secret shares. If it does, then it is very likely that these shares are indeed from player i. They will also check Eq. 2 to validate the complaint. If it is valid, all honest users will reconstruct a_i by broadcasting their shares. This is the straightforward process of solving a set of linear equations that specify the internal secret for that player. The number of shares required for reconstruction depends on the structure of E and a_i.

Using the private key Based on the definition of x, it suffices to find a. We can find a directly through the relation $aE = (x_1, x_2, \ldots, x_n)$ if we know a sufficient fraction of the x_i. This fraction is a function of the structure of E. It is desirable, however, to never reveal a. Note that since a is a linear function of the x_i we can write $a = (x_1, x_2, \ldots, x_n)\Sigma$, where Σ is the pseudo-inverse of E. If $T(a) = av$ then $T(a) = (x_1, x_2, \ldots, x_n)\Sigma v$. If the signature or encryption is of the form g^x, then each player can sign g^{x_i} and the signatures or encryptions are combined as $\prod g^{x_i L_i}$, where L_i is the ith row of Σv. Of course, if only a subset of the x_i are available, a different Σ must be used.

3.4 GJKR DKG

Before diving into sparse matrix DKG, we will show how the algorithm introduced by Gennaro, Jarecki, Krawczyk, and Rabin fits into the more generic matrix evaluated DKG framework. This protocol was introduced to eliminate a security flaw in the simpler Joint-Feldman protocol proposed by Pedersen where an adversary could bias the distribution over public keys.

Let E be the Vandermonde matrix with $m \triangleq t + 1$ rows and n columns, where t is the security threshold of the system:

$$E_{ij} = j^{i-1}$$

Each a_i is a random m element column vector so that $|Q_i| = n$. The matrix makes the external secret, s_i, the same as evaluations of a polynomial (with coefficients defined by the internal secret) at the points $1, 2, \ldots, n$.

The private key is just a function applied to the sum of the internal secrets of the valid players. The function in this case is $T(b_1, b_2, \ldots, b_m) = b_1$. Each player's share of this secret is the sum of the shares they hold from the other valid players. Equivalently, each player's share is the evaluation at a certain point of the polynomial created by summing the polynomials of the valid players. Since all the polynomials are of degree t, any $t+1$ of the $\sum_{i \in V} s_{ij}$ will allow interpolation to reconstruct $\sum_{i \in V} a_i$. Hence with $t+1$ valid shares, the algorithm can succeed in revealing the public or private key. Of course, since the private key should never be revealed, it is possible to use the key using just the evaluations of the polynomial through Lagrange interpolation coefficients.

Assuming that each server behaves properly, this algorithm incurs communication cost per player of $O(t)$ broadcast bits and $O(n)$ messages, assuming no faulty players. Achieving the broadcast through Byzantine agreement is very expensive. In the the two phases of the algorithm, each server performs $O(tn)$ operations. This occurs because each server must check every other server, and checking each server is an $O(t)$ operation.

4 Sparse Matrix DKG

To reduce both communication cost and computational complexity, we introduce a technique that that relies on a sparse evaluation matrix. This produces a $(\gamma - \epsilon, \gamma + \epsilon, \delta)$ probabilistic threshold DKG algorithm for $0 < \gamma < \frac{1}{2}, \epsilon > 0, \delta > 0$.

4.1 Intuition

The basic algorithm only requires a user to communicate with her checking group (to share her secret) and with all the checking groups to which she belongs (to determine whether those users behaved as the protocol dictates). To reduce the communication complexity of the algorithm it is sufficient to make these checking groups small. However, they must be sufficiently large so that a majority of the group is honest. If the groups are logarithmic in size, Hoeffding bounds give an inverse-polynomial probability of a group failing this criteria, assuming that the fraction of honest players is greater than a half. For recovery to be possible, the internal secret of a player must have fewer degrees of freedom than the number of honest recipients of her shares. A sparse, structured evaluation matrix is the tool that we use to map small secrets onto small checking groups.

4.2 Sparse Evaluation Matrix

By imposing constraints on the number of non-zero entries in each user's internal secret and having a very structured evaluation matrix, we reduce the cost of the problem from quadratic to polylogarithmic in the number of users. Figure 1

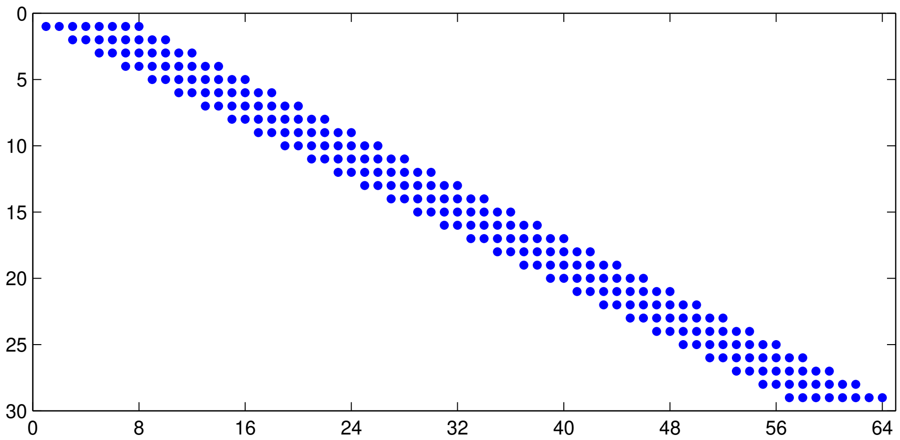

Figure 1. An illustration of an evaluation matrix for $\gamma = \frac{1}{2}$. Here $n = 64, \ell = 8$. Each dot indicates a non-zero position in the matrix.

illustrates the evaluation matrix used for an $\left(\frac{1}{2} - \epsilon, \frac{1}{2} + \epsilon, \delta\right)$ instance of the algorithm. Each row of the evaluation matrix has ℓ consecutive, random entries offset by two columns from the previous row. Since the matrix must have n columns, it must also have $m \triangleq (n - \ell)/2 + 1$ rows. We will show that for $\ell = O(\log n)$, the algorithm can recover the key with a $\frac{1}{2} + \epsilon$ fraction of honest shares with high probability. In general, for $\gamma \leq \frac{1}{2}$, the evaluation matrix E has the band in each row offset by γ^{-1} columns (on average) from the previous row and hence there are $m \approx \gamma(n - \ell)$ rows in E. Intuitively, this offset allows most sets of γn columns of the matrix to have full rank. The rest of the description of the algorithm and the accompanying proofs assume $\gamma = \frac{1}{2}$.

4.3 Dealing Phase

The dealer must generate $O(n \log n)$ independent random bits for the analysis of this algorithm to succeed. This randomness is used to determine two things. First, it establishes the evaluation matrix E. The second use is to create a permutation of the users. All aspects of the algorithm require the adversary and honest players to be distributed at random. Any large cluster of dishonest players leaves part of the private key vulnerable.

Unfortunately, it isn't practical to distribute that many bits to all the players. It is reasonable, however, for the dealer to send each player his identifier as determined by the permutation. Then player i only needs to know the columns Q_i and $\bigcup_{j|i \in Q_j} Q_j$ of E. The first group is needed to produce the shares to send to other players and the second, larger group is needed to check and possibly reconstruct other players' shares. As we will see, this is a logarithmic number of columns, and hence a polylogarithmic number of random bits. All players will also query the dealer for the addresses of their peers. Since the dealer is assumed to be honest, these bits need not be "broadcast"; a simple point-to-point distribution suffices.

4.4 Making and sharing the secret

Player i creates a (sparse) internal secret vector, a_i that contains only $u = \ell/2\epsilon^2$ non-zero, consecutive elements. The position of these elements is chosen to equally distribute the checking load among all the players. Specifically, player i will have her terms start at $\lceil i\frac{m-u}{n}\rceil$. The structure of both the internal secret vectors and the evaluation matrix mean that not all players are given non-zero shares by player i. This defines the checking group, Q_i. The only recipients of broadcasts regarding player i are members of her checking group. In this setting of γ, there will be $\ell + 2(u-1)$ (non-zero) shares, so that an adversary who controls any $1/2 - \epsilon/2$ fraction of these still has fewer than u shares.

To prove that these shares were constructed from the same internal secret vector, it is necessary for player i to broadcast the C_{ij} as defined above to Q_i. Since our system model uses Byzantine agreement to achieve broadcast, the relatively small number of non-zero shares reduces the number of participants in this operation. In this algorithm, since the set of players outside of Q_i has no evidence as to whether player i is or is not valid, the other players will take a majority vote of the checking group to determine that. We are guaranteed that all the honest players in the checking group will agree whether player i is valid since they are privy to the same information. We must then show that a majority of Q_i is honest. This is a straightforward application of Hoeffding bounds since the expected fraction of honest players in any group is $\gamma + \epsilon$:

$$\begin{aligned}\Pr\left(|\{x \in Q_i : x \text{ is dishonest}\}| > \left(\gamma - \frac{\epsilon}{2}\right)|Q_i|\right) &\leq 2\exp\left(-\frac{(|Q_i|\epsilon/2)^2}{2|Q_i|}\right)\\ &= 2\exp\left(-\frac{|Q_i|\epsilon^2}{8}\right)\end{aligned}$$

Hence, since $|Q_i| > \frac{\ell}{\epsilon^2}$, if $\ell > O((1+\kappa)\log n)$, then the probability is bounded by $n^{-1-\kappa}$ and the probability that all checking groups have a majority honest players is $1 - n^{-\kappa}$ by a union bound.

4.5 Extracting the public key

Revealing the public key occurs just as in the more general description. In the proof of correctness, we show that it happens with high probability. A third party who wishes to discover the public key that the algorithm has generated needs to ask each checking group for the checked user's share and take the majority (the checking group may also respond that the player is invalid). This is an $O(n \log n)$ operation. It is reasonable to assume that the dealer poll each player about the other players that they have checked, tally these votes for each player, multiply the public key shares for the valid players and then send the public key to every player.

It may be desirable to do this operation in a more distributed fashion. It is certainly not practical for each player to poll each checking group. Instead, a message passing algorithm to tally the public key is feasible. Here we set up a binary partition of all the players. At the bottom of a binary tree are small[2], non-overlapping groups of users for which the public key share of each (valid) member is known to all in the group. As one moves up the tree, these shares are multiplied to establish the shares of larger groups. The global public key is found at the top of the tree. A logarithmic sized group is responsible for each aggregation point within the binary tree. The public key, once found, may be passed back down the tree to each user. By restricting communication to the edges of the tree, each player need only communicate with $O(\log n)$ other players.

4.6 Using the global secret

As described above, the global secret is maintained as shares kept by each of the valid users. However, to be useful in threshold cryptosystems, it must be possible to apply this key for a signature without having all shareholders act. The structure of the evaluation matrix assists us in this regard. Discovering the value of the private key requires the solving for a (implicitly). If we can find the inversion matrix Σ, we will be able to recover the private key. This is possible if the submatrix of the evaluation matrix corresponding to the valid users has rank m.

4.7 Recoverability

We assume that there are βn players ($\beta \triangleq \gamma + \epsilon$) that will contribute their shares to sign (or decrypt) a message. Let S be the set of these players. Our goal is to show that $E|_S$, the $m \times |S|$ submatrix of E consisting of the columns corresponding to good shares has rank m. We first state a lemma that concerns biased random walks. In the appendix, we prove this lemma for $\gamma = \frac{1}{2}$. It is straightforward to extend this to arbitrary γ.

Lemma 1. *An n step random walk that reflects at zero, with $(\gamma + \epsilon)n$ steps of $1 - \gamma^{-1}$ and $(1 - \gamma - \epsilon)n$ steps of $+1$, will reach $O(\kappa \log n)$ with probability less than $n^{-\kappa}$, for fixed ϵ.*

We can now prove our theorem concerning the rank of $E|_S$, in the case that $\gamma = \frac{1}{2}$.

Theorem 1. *The matrix $E|_S$ formed by randomly deleting αn columns from E, $\alpha = \frac{1}{2} - \epsilon$, will have rank m with probability $1 - n^{-\kappa}$ if each row of E has $\ell = -\frac{(2+\kappa)}{\log(1-2\epsilon)} \log n$ consecutive, non-zero, random entries.*

[2] It is important that these groups be large enough that there be a low probability of a corrupt majority. It suffices for them to be of size $\Omega(\log n)$.

Proof. The theorem is proved by showing that a subset of m columns of $E|_S$, taken as a matrix $E|'_S$, has random elements along its main diagonal. Consider the process adding columns from E to $E|_S$. Each row of E has ℓ non-zero, consecutive entries, called the band. We consider the incremental process of examining columns in order. If a column is present, and the column is not to the left of the band for the current row being considered (i.e., the row r for which we want a random value at $(E'_S)_{rr}$), that column is added to $E|'_S$, and the next row is considered. Let X_i denote the offset of column i in the non-zero entries of the currently considered row, R_i (where i is a column index for the full matrix, E). For example, in Fig. 1, if $i = 8$ for row 3 the offset is 4. In general, $X_i = i - 2(R_i - 1)$. If X_i ever exceeds ℓ, the process fails since none of the later columns will be able to contribute a random element for the diagonal entry at the current row. Define $X_1 = 1, R_1 = 1$. Now consider the state X_i. If column i is missing, we stay at the current row, $R_{i+1} = R_i$, but step forward in relation to the beginning of the band. Hence $X_{i+1} = X_i + 1$. Now if column i is present, there are two possibilities. If $X_i \geq 1$, the column is added and $R_{i+1} = R_i + 1$, so that column $i + 1$ would be one step forward in the same row, but one step behind in the next row (since the rows are offset by two relative to each other), and $X_{i+1} = X_i - 1$. Now if $X_i = 0$ a present column does not help since we are in front of the band, so $X_{i+1} = 0, R_{i+1} = R_i$.

Observing just the X_i, the process is identical to the reflecting random walk process defined above. Applying the random walk lemma, we may bound the probability that the walk passes t after n steps, one for each column of A. This probability is just:

$$P(\text{process fails}) = P(\text{walk passes } \ell \text{ in } n \text{ or fewer steps}) < n^{-\kappa}$$

4.8 Correctness

Theorem 2. *The sparse matrix DKG algorithm described above is a probabilistic threshold* $\left(\frac{1}{2} - \epsilon, \frac{1}{2} + \epsilon, 2n^{-\kappa}\right)$ *DKG algorithm for* $\ell = O(\kappa \log n)$, *the width of the band in the constraint matrix.*

Proof. (C1) The honest players within checking group Q_i always agree whether player i is in V, since that decision is made based on broadcast information. Honest players outside the checking group will rely on a majority vote of the checking group to determine if player i is included in V. Then the set V that is established in step 3 of the algorithm is unique if each checking group has a majority of honest players, which happens with probability $1 - n^{-\kappa}$. Assuming the adversary is not able to compute $\mathrm{dlog}_g h$, the check in Eq. 1 implies that the player's shares are all consistent with the same a_i. The reconstruction theorem implies that the honest players for any set of rows will have full rank, so they are consistent with a unique a_i, with probability $1 - n^{-\kappa}$. Since all honest players have shares that satisfy Eq. 1, any set of them are able to recover x or not recover anything (in the case that the submatrix that they define is not invertible).

(C2) In the case where a valid complaint is filed against player i, this a_i is reconstructed in public using the reconstruction theorem and $A_i = g^{a_i} \mod p$.

Now we consider the case where no valid complaints are filed. Since a_i could have been reconstructed with the shares from the honest players, there are at least u linearly independent equations and u unknowns, so that there must be a unique solution. Since the broadcast from player i agrees with these equations, through Eq. 2, the broadcast A_i is exactly g^{a_i}. Hence all honest players within each checking group have the correct value of $A_i, i \in V$, and since there are a majority of honest players in each checking group, all honest players have the same value for $A_i, i \in V$ and also $y = g^x \mod p$.

(C3) The secret is defined as $x = \left(\sum_{i \in V} a_i\right) v$. Since v is random, any random term of some a_i that is independent from the other a_j will cause x to be uniform. In the proof of secrecy, we show that the adversary cannot determine any a_i completely for an honest player i, so the dishonest players' a_j is independent of a term of every honest a_i. Also honest players choose all their terms independently, so the a_i from any honest player will satisfy this requirement.

(C4) This is a direct application of the reconstruction theorem.

4.9 Secrecy

To show that the adversary is not able to learn any information about the private key x other than the fact that it is the discrete log of the public key y, we create a simulator. Formally, a *simulator* is a probabilistic polynomial-time algorithm that given $y \in Z_p$, such that $y = g^x \mod p$ for some x, can produce a distribution of messages that is indistinguishable from a normal run of the protocol where the players controlled by the simulator are controlled instead by honest players. This is the familiar technique used to show that zero-knowledge proofs do not reveal any private information. However, since our algorithm relies on a random distribution of adversarially controlled players, our simulator will only have a high probability of success.

We assume that the adversary controls no more than a $\frac{1}{2} - \epsilon$ fraction of the players, chosen before the start of the algorithm. Recall that in section 4.4 we required that the adversary control no more than a $\frac{1-\epsilon}{2}$ fraction of any checking group. Hence each checking group contains less than $\frac{1-\epsilon}{2}\left(\ell + 2\left(\frac{\ell}{2\epsilon^2} - 1\right)\right) < \frac{\ell}{2\epsilon^2} < u$ adversarially controlled players and the adversary cannot learn an honest player's entire internal secret.

The input to the simulator is a y that could have been established at the end of a normal run of the protocol. Assume that the adversary controls the set $\mathcal{B} = \{B_1, B_2, \ldots, B_t\}$ and that the honest parties (controlled by the simulator) are $\mathcal{G} = \{G_1, G_2, \ldots, G_{n-t}\}$. In a non-simulated run of the protocol that ends with the public key y, the a'_i are uniform random vectors, subject to the sparseness constraint, and the a_i are random vectors subject to both $\prod_{i \in V} \prod_j g^{a_{ij} v_j} = y$ and the sparseness constraint.

Consider the following algorithm for the simulator, SIM:

1. Each honest player $i \in \mathcal{G}$ performs steps 2 and 3 of the sparse matrix DKG protocol. At this point:
   - The set V is well defined and $\mathcal{G} \subseteq V$.

– The adversary $\mathcal{B}$ has seen a_i, a'_i for $i \in \mathcal{B}$, s_{ij}, s'_{ij} for $i \in V, j \in \mathcal{B}$ and C_{ij} for $i \in V$.
– SIM knows a_i, a'_i for all $i \in V$ (including those in $V \cap \mathcal{B}$, the internal secrets for the consistent adversary players).

2. Perform the following calculations:
   – Compute $A_i \triangleq g^{a_i} \mod p$ for $i \in V \setminus \{G_1\}$.
   – Let S be a subset of Q_{G_1}, the checking group for G_1, that contains all of $Q_{G_1} \cap \mathcal{B}$ and enough of $Q_{G_1} \cap \mathcal{G}$ so that the rank of $E|_S$, the columns of E corresponding to S, has rank $u-1$. Let r be some i such that the columns $S \cup \{r\}$ have rank u.
   – Assign $s^*_{G_1 j} \triangleq s_{G_1 j}$ for $j \in S$.
   – Let S' be a subset of $u-1$ elements of S such that the columns S' of E have rank $u-1$.
   – Compute Σ, the inverse of the submatrix of E corresponding to the columns $S' \cup \{r\}$ and the rows $1, \ldots, u$ (i.e., assume wlog that $G_1 = 1$).
   – Note that $a^*_{G_1} = (s^*_{G_1 S'_1}, \ldots, s^*_{G_1 S'_{u-1}}, s^*_{G_1 r})\Sigma$, but $s^*_{G_1 r}$ has not yet been fixed. Similarly $A^*_{G_1} = (g^{\alpha_1 + \beta_1 s^*_{G_1 r}}, \ldots, g^{\alpha_u + \beta_u s^*_{G_1 r}})$, where the α_i, β_i are functions of Σ and $s^*_{G_1 j}, j \in S'$.
   – For our construction to succeed, $\prod_j A^{v_j}_{G_i j} = y \left(\prod_{i \in V \setminus \{G_1\}} \prod_j A^{v_j}_{ij}\right)^{-1}$. The right hand side is known and of the form g^{γ}, and the left hand side is of the form $g^{\alpha + \beta s^*_{G_1 r}}$. Hence we can solve to find $g^{s^*_{G_1 r}}$ and then evaluate the expression for $A^*_{G_1}$.
3. Broadcast A_i for $i \in \mathcal{G} \setminus G_1$ and $A^*_{G_1}$.
4. Perform the checks of step 4.(b) of the algorithm for each player $i \in \mathcal{G}$ on the $A_j, j \in \mathcal{B}$ broadcast by the adversary's players and broadcast any necessary complaints.
5. The adversary cannot file a valid complaint against any honest player since all the messages followed the protocol. However, the simulator must recover from the adversary's actions. The simulator will follow the protocol of 4.(c) to recover the a_i for players who did not share their A_i properly.

The simulator will result in an identical distribution of the observed a_i, a'_i as would be expected from a non-simulated run that generated the public key y. This is because the simulator uses the original, random a_i, a'_i for $i \in \mathcal{G} \setminus G_1$. Also $a^*_{G_1}$ is chosen to be random subject to the constraint above. It remains to be shown that $a'^*_{G_1}$ is also random. The relationship between $a^*_{G_1}, a'^*_i$ is established through the public $C_{G_1} = g^{a^*_{G_1}} h^{a'^*_{G_1}}$. This implies $a'^*_{G_1} = \mathrm{dlog}_g(h)^{-1} \cdot (a_{G_1} - a^*_{G_1}) + a'_{G_1}$, which inherits its randomness from a'_{G_1}.

4.10 Communication Complexity and Running Time

The primary objective of the matrix-based constraint algorithm is to reduce both the communication complexity and running time for the users participating in the key generation without significantly affecting the chance that the algorithm fails to properly generate a key. As described above, a fraction of each user's secret as well as a bit-committed version of the secret will be sent to $\ell/2\epsilon^2 = O(\log n)$ other users. Hence this communication accounts for $O(\ell^2)$ messages since the secret is of length $O(\ell)$, using Byzantine agreement for the broadcast. If $\gamma > \frac{1}{3}$, then we must use authenticated Byzantine agreement [6]. Also, we will incur greater costs in the dealing phase since this operation cannot be simply composed [7]. If $\gamma < \frac{1}{3}$ we can reduce the dealing cost by using the technique proposed by Cachin, Kursawe and Shoup [3]. In the presence of dishonest players, this cost grows by a factor of $O(\ell)$ since each dishonest player can cause two more broadcasts to occur within the checking group. In the second phase of the protocol, without any adversaries the cost is again $O(\ell^2)$ messages. With adversaries, this cost again increases by a factor of $O(\ell)$ since each member of the checking group must broadcast.

The running time for this algorithm is also much shorter than that of previous solutions to this problem. Each player is a member of $O(\ell)$ checking groups and must check one equation for each. Each equation is a product of $O(\ell)$ modular exponentiations, so the cost is $O(\ell^2)$ exponentiations.

The constants in these asymptotic expressions are very reasonable. For an n^{-2} chance of failure, if $\epsilon = 1/10$, then a suitable setting for ℓ is $17 \log n$. A linear increase of ℓ results in either an exponentially smaller failure probability or a linear decrease in ϵ. Since the checking groups are of size $\frac{\ell}{2\epsilon^2}$, they are more sensitive to the value of ϵ. In practice, it is reasonable for the gap between the the fraction of dishonest parties and the fraction of shares required for reconstruction to be a fixed constant, so the size of the Q_i is logarithmic with a small leading constant.

References

1. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the First ACM Conference on Computer and Communications Security, ACM Press (1993) 62–73
2. Blakley, G.R.: Safeguarding cryptographic keys. **48** (1979) 313–317
3. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantipole: Practical asynchronous byzantine agreement using cryptography (extended abstract). In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, ACM Press (2000) 123–132
4. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Lecture Notes in Computer Science **1592** (1999) 295+
5. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Revisiting the distributed key generation for discrete-log based cryptosystems. (2003)
6. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) **4** (1982) 382–401
7. Lindell, Y., Lysyanskaya, A., Rabin, T.: On the composition of authenticated byzantine agreement. In: Proceedings of the Thirty-fourth Annual ACM Symposium on Theory of Computing, ACM Press (2002) 514–523

8. Pedersen, T.P.: A threshold cryptosystem without a trusted party. Lecture Notes in Computer Science **547** (1991) 522–526
9. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. Lecture Notes in Computer Science **576** (1991) 129–140

5 Appendix

Lemma 2. *Consider a (reflecting) random walk X_i defined in terms of a sequence of differences D_i:*

$$\begin{aligned} X_0 &= 0 \\ X_{i+1} &= \max(0, X_i + D_{i+1}) \end{aligned}$$

The sequence of differences $D_1, D_2, \ldots, D_n$, is generated at random and satisfies:

$$|\{i|D_i = 1\}| = \alpha n \triangleq r, |\{i|D_i = -1\}| = (1-\alpha)n \triangleq s$$

Let $\alpha \triangleq 1/2 - \epsilon$. Then the probability that the walk has reached $\ell \triangleq -\frac{(2+\kappa)}{\log(1-2\epsilon)} \log n$ in n or fewer steps is $P(X_j = \ell, j \leq n) < n^{-\kappa}$.

Proof. Let $B_{i,j}$ be the event that $X_{i+j} = \ell, X_i = 0, X_k \neq 0, i < k < i+j$. That is the event where i is the last time that the walk is at 0 and the walk is at ℓ after j more steps. Of those j steps, such a walk will have exactly ℓ more steps to the right than steps to the left. Hence there are $d \triangleq \frac{j-\ell}{2}$ left steps and $d+\ell$ right steps. There are fewer than $\binom{j}{d}$ ways to choose the order of the steps. Condition the sequence of differences on those j steps:

$$\begin{aligned} P(B_{i,j}) &< \binom{j}{d} \frac{\left(\prod_{k=0}^{d-1}(s-k)\right)\left(\prod_{k=0}^{d+\ell-1}(r-k)\right)}{\prod_{k=0}^{j-1}(n-k)} \\ &= \binom{j}{d} \prod_{k=0}^{d-1} \frac{(s-k)(r-k)}{(n-2k)(n-2k-1)} \prod_{k=0}^{\ell-1} \frac{r-d-k}{n-2d-k} \end{aligned}$$

Observe that if $a+b=c$ and $b < \frac{c}{2}-1$ then $\frac{ab}{c(c-1)} < \frac{1}{4}$. Also $\frac{r-d-k}{n-2d-k} < \frac{r}{n} = \frac{1}{2} - \epsilon$, for arbitrary d, k since $r < \frac{n}{2}$ so that:

$$\begin{aligned} P(B_{i,j}) &< \binom{j}{d} \left(\frac{1}{4}\right)^{\frac{j-\ell}{2}} \left(\frac{1}{2} - \epsilon\right)^{\ell} \\ &= \binom{j}{d} \left(\frac{1}{2}\right)^{j} (1-2\epsilon)^{\ell} \\ &< (1-2\epsilon)^{\ell} \end{aligned}$$

So $P(B_{i,j}) < (1-2\epsilon)^{\ell} = n^{-(2+\kappa)}$. Now there are fewer than n choices for either i or j so that $P(\cup_{i,j} B_{i,j}) < n^2 n^{-(2+\kappa)} = n^{-\kappa}$.