Torus DKG Technical Specification

Precursors

- (Optional) A Practical Scheme for Non-interactive Verifiable Secret Sharing (aka Feldman's VSS) by Feldman (1987)

- (Optional) A Threshold Cryptosystem Without A Trusted Party (aka Pedersen's DKG) by Pedersen (1991)
  - Uses N runs of Feldman VSS to create a DKG

- (Optional) Secure Distributed Key Generation for Discrete-Log Based Cryptosystems by Gennaro et al. (2006)
  - Showed that Joint-Feldman DKG (simplified version of Pedersen's DKG) resulted in non-uniform outputs in the shared secret.
  - Introduced Secure-DKG, which uses a Pedersen-VSS instead of Feldman's VSS.
    - This results in the polynomial commitment being a Pedersen commitment instead of a DLog commiment
      - Final commitment polynomial has a form $g^{s_0}h^{s'_0} + g^{s_1}h^{s'_1} + g^{s_2}h^{s'_2}$ ...
      - where g^{s_0} is the public key and s_0 is the shared secret (private key)
    - Introduced a way to "extract" g^{s_0} from $g^{s_0}h^{s'_0}$ (Fig 2. Protocol DKG - Step 4)

- Asynchronous Verifiable Secret Sharing (AVSS) by Cachin et al. (2002) - Cachin2002
  - Asynchronous network model
  - Introduced AVSS, a verifiable secret sharing Byzantine-tolerant protocol
  - Introduced asynchronous Proactive Secret Sharing (PSS)
    - Uses AVSS as a subprotocol
    - Uses Multi-Valued Validated Byzantine Agreement (MVVBA) to select which set of AVSS to be used in refresh protocol

- Distributed Key Generation in the Wild by A Kate, Y Huang, I Goldberg (2012) - Kate2012
  - Asynchronous network model
  - Introduced HybridDKG, a DKG protocol built on top of AVSS
    - Uses AVSS as a secret-sharing subprotocol
    - Avoids use of multi-valued validated byzantine agreement

    - By using leader-initiated reliable broadcast with leader change
    - Results in optimistic / pessimistic operation phase
  - Introduced Discrete Log Equality (DLEQ) NIZKP (Section 5.3) that proves equality of s_0 in $g^{s_0}h^{s_0'}$ and g^{s_0}, replacing the "extract" protocol in Protocol DKG - Step 4

Introduction

The Torus Network's main purpose is to generate distributed shares of private keys across nodes using distributed key generation (DKG). We have made slight modifications to HybridDKG using results from AVSS, and the exact implementation is given below in the overview.

The Torus Network is also able to migrate shares across dynamic committees through proactive secret sharing (PSS). However, this is but a result of Cachin2002, and no modifications were made to the Refresh protocol from that paper.

Overview of Torus DKG

Similar to HybridDKG from Kate2012, Torus DKG builds a DKG protocol on top of AVSS from Cachin2002, by using AVSS as a secret-sharing subprotocol. The main difference between Torus DKG and HybridDKG is that we *do* use a multi-valued validated byzantine agreement protocol - Tendermint. This also means Torus DKG only has partial synchrony assumptions since Tendermint only has partial synchrony assumptions.

For the Torus DKG, we have n nodes that run n independent rounds of AVSS. We then use MVVBA (Tendermint) to select a threshold set of AVSS as the qualified set. We then use the DLEQ NIZKP from Kate2012 to "extract" the public key from the Pedersen polynomial commitments.

We assume $n - t \geq k > t$, where t is the number of nodes controlled by the adversary, k is the reconstruction threshold, and n is the total number of nodes.

The Torus DKG satisfies the following conditions for any t-limited adversary:

- Liveness: Honest nodes are able to complete the DKG protocol, except with negligible probability
- Correctness: If at least k honest nodes have completed a DKG, they hold a sharing of s_0, except with negligible probability
- Secrecy: An adversary set $\{S_i\}$ of at most t shares cannot compute s_0 except with negligible probability
- Efficiency: For every ID, the communication complexity of DKG instance ID is uniformly bounded

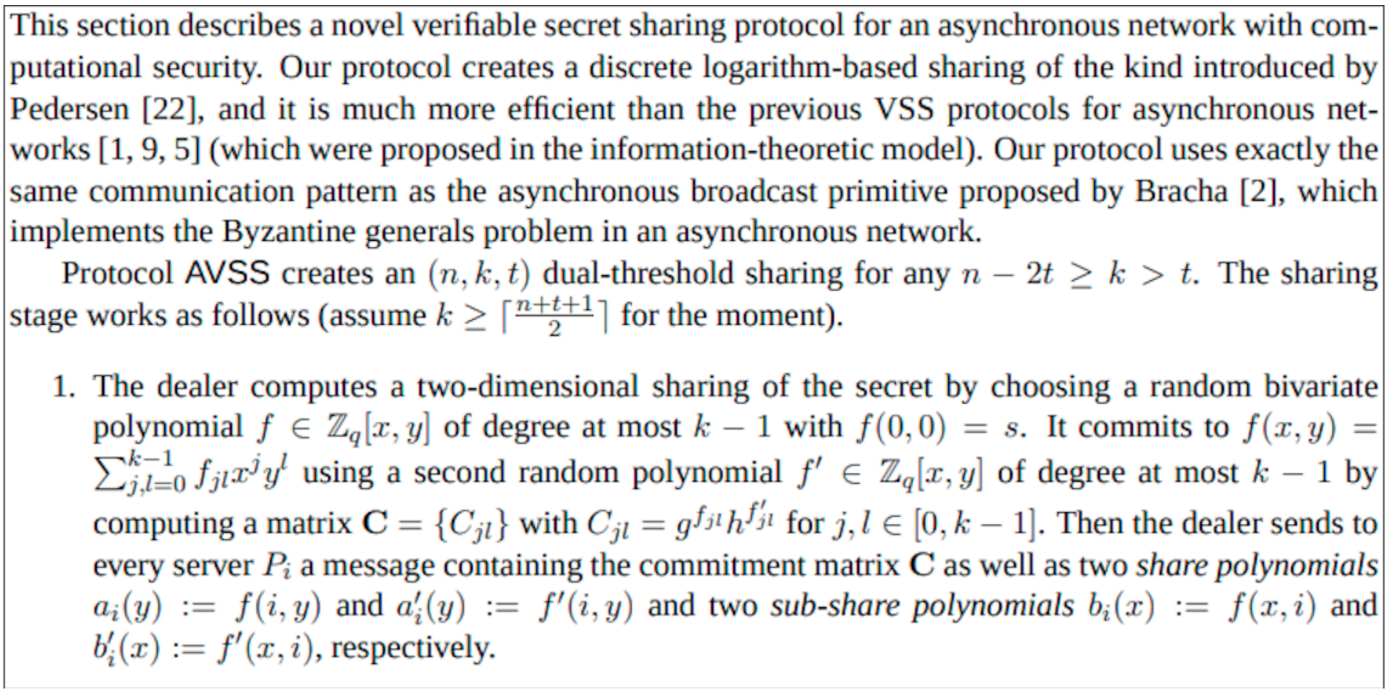
We first cover the subprotocols used in the Torus DKG - MVVBA, AVSS and the NIZKPK DLEQ proof.

Multi-Valued Validated Byzantine Agreement

Byzantine agreement implements a binary decision that guarantees a particular outcome only if all honest nodes propose the same value. Validated byzantine agreement extends this to an external global, polynomial-time computable predicate known to all nodes. Multi-valued validated byzantine agreement is a protocol that is able to do this for multiple values. In Cachin2002 he references an MVVBA protocol with asynchronous network assumptions, but we use Tendermint instead, with partial synchrony network assumption.

MVVBA is used specifically in the DKG protocol to select a proposal of a threshold set of sharings that are complete / will complete. We make use of Tendermint's deterministic state machine algorithm to decide on a single valid proposal. Since the predicate (verify-termination, defined later) is polynomial-time computable, assuming that Tendermint is Byzantine fault tolerant up to a threshold number of nodes, this satisfies the requirement for a MVVBA protocol.

AVSS (Cachin2002)


This section describes a novel verifiable secret sharing protocol for an asynchronous network with computational security. Our protocol creates a discrete logarithm-based sharing of the kind introduced by Pedersen [22], and it is much more efficient than the previous VSS protocols for asynchronous networks [1, 9, 5] (which were proposed in the information-theoretic model). Our protocol uses exactly the same communication pattern as the asynchronous broadcast primitive proposed by Bracha [2], which implements the Byzantine generals problem in an asynchronous network.

Protocol AVSS creates an (n, k, t) dual-threshold sharing for any $n - 2t \geq k > t$. The sharing stage works as follows (assume $k \geq \lceil \frac{n+t+1}{2} \rceil$ for the moment).

1. The dealer computes a two-dimensional sharing of the secret by choosing a random bivariate polynomial $f \in \mathbb{Z}_q[x, y]$ of degree at most $k - 1$ with $f(0, 0) = s$. It commits to $f(x, y) = \sum_{j,l=0}^{k-1} f_{jl} x^j y^l$ using a second random polynomial $f' \in \mathbb{Z}_q[x, y]$ of degree at most $k - 1$ by computing a matrix $\mathbf{C} = \{C_{jl}\}$ with $C_{jl} = g^{f_{jl}} h^{f'_{jl}}$ for $j, l \in [0, k - 1]$. Then the dealer sends to every server P_i a message containing the commitment matrix $\mathbf{C}$ as well as two *share polynomials* $a_i(y) := f(i, y)$ and $a'_i(y) := f'(i, y)$ and two *sub-share polynomials* $b_i(x) := f(x, i)$ and $b'_i(x) := f'(x, i)$, respectively.

2. When they receive the send message from the dealer, the servers *echo* the points in which their share and sub-share polynomials overlap to each other. To this effect, P_i sends an echo message containing $\mathbf{C}$, $a_i(j)$, $a'_i(j)$, $b_i(j)$, and $b'_i(j)$ to every server P_j.

3. Upon receiving k echo messages that agree on $\mathbf{C}$ and contain valid points, every server P_i interpolates its own share and sub-share polynomials $\bar{a}_i, \bar{a}'_i, \bar{b}_i$, and $\bar{b}'_i$ from the received points using standard Lagrange interpolation. (In case the dealer is honest, the resulting polynomials are the same as those in the send message.) Then P_i sends a ready message containing $\mathbf{C}, \bar{a}_i(j), \bar{a}'_i(j)$, $\bar{b}_i(j)$, and $\bar{b}'_i(j)$ to every server P_j.

   It is also possible that a server receives k valid ready messages that agree on $\mathbf{C}$ and contain valid points, but has not yet received k valid echo messages. In this case, the server interpolates its share and sub-share polynomials from the ready messages and sends its own ready message to all servers as above.

4. Once a server receives a total of $k+t$ ready messages that agree on $\mathbf{C}$, it *completes* the sharing. Its share of the secret is $(s_i, s'_i) = (\bar{a}_i(0), \bar{a}'_i(0))$.

The reconstruction stage is straightforward. Every server P_i reveals its share (s_i, s'_i) to every other server, and waits for k such shares from other servers that are consistent with the commitments $\mathbf{C}$. Then it interpolates the secret $f(0,0)$ from the shares.

For smaller values of k, in particular for $t < k < \lceil \frac{n+t+1}{2} \rceil$, the protocol has to be modified to receive $\lceil \frac{n+t+1}{2} \rceil$ echo messages in step 3. This guarantees the uniqueness of the shared value.

A detailed description of the protocol is given in Figures 1 and 2. In the protocol description, the following predicates are used:

verify-poly$(\mathbf{C}, i, a, a', b, b')$, where a, a', b, and b' are polynomials of degree $k-1$, i.e.,

$$a(y) = \sum_{l=0}^{k-1} a_l y^l, \qquad a'(y) = \sum_{l=0}^{k-1} a'_l y^l, \qquad b(x) = \sum_{j=0}^{k-1} b_j x^j, \quad \text{and} \quad b'(x) = \sum_{j=0}^{k-1} b'_j x^j;$$

the predicate verifies that the given polynomials are share and sub-share polynomials for P_i consistent with $\mathbf{C}$; it is true if and only if for all $l \in [0, k-1]$, it holds $g^{a_l} h^{a'_l} = \prod_{j=0}^{k-1} (C_{jl})^{i^j}$, and for all $j \in [0, k-1]$, it holds $g^{b_j} h^{b'_j} = \prod_{l=0}^{k-1} (C_{jl})^{i^l}$.

verify-point$(\mathbf{C}, i, m, \alpha, \alpha', \beta, \beta')$ verifies that the given values α, α', β, and β' correspond to the points $f(m,i)$, $f'(m,i)$, $f(i,m)$, and $f'(i,m)$, respectively, committed to in $\mathbf{C}$, which P_i supposedly receives from P_m; it is true if and only if $g^{\alpha} h^{\alpha'} = \prod_{j,l=0}^{k-1} (C_{jl})^{m^j i^l}$ and $g^{\beta} h^{\beta'} = \prod_{j,l=0}^{k-1} (C_{jl})^{i^j m^l}$.

verify-share$(\mathbf{C}, m, \sigma, \sigma')$ verifies that the pair (σ, σ') forms a valid share of P_m with respect to $\mathbf{C}$; it is true if and only if $g^{\sigma} h^{\sigma'} = \prod_{j=0}^{k-1} (C_{j0})^{m^j}$.

The servers may need to interpolate a polynomial a of degree at most $k-1$ over $\mathbb{Z}_q$ from a set $\mathcal{A}$ of k points $\{(m_1, \alpha_{m_1}), \ldots, (m_k, \alpha_{m_k})\}$ such that $a(m_j) = \alpha_{m_j}$ for $j \in [1, k]$. This can be done using standard Lagrange interpolation. We abbreviate this by saying a server *interpolates* a from $\mathcal{A}$; should $\mathcal{A}$ contain more than k elements, an arbitrary subset of k elements is used for interpolation.

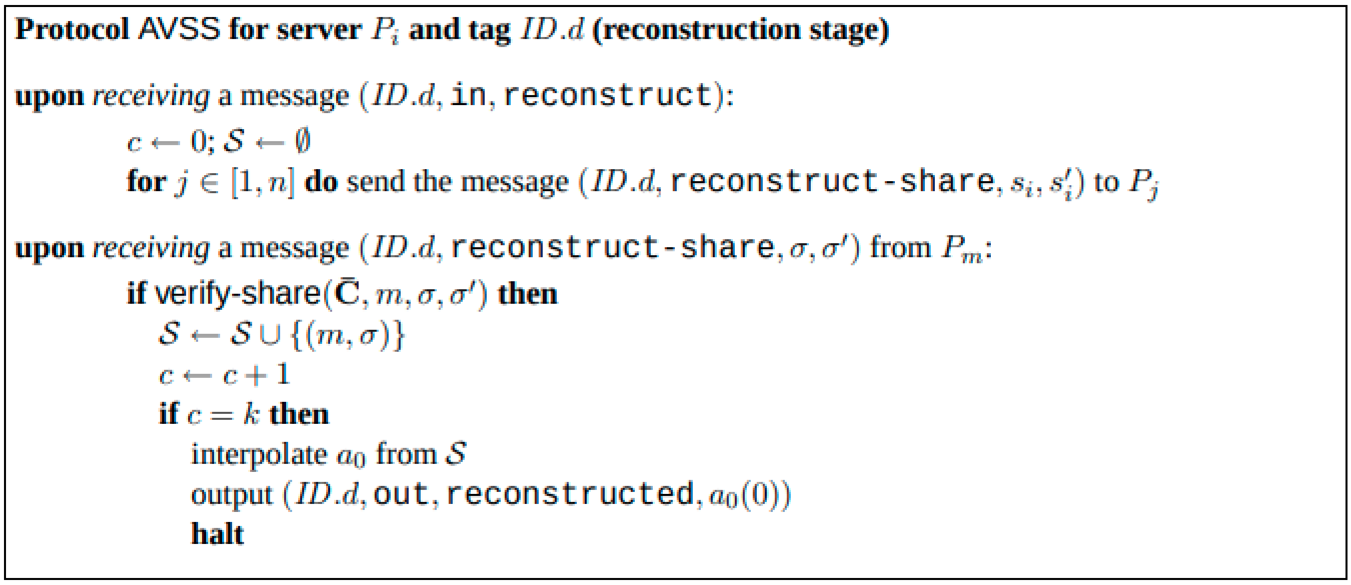
In the protocol description, the variables e and r count the number of echo and ready messages, respectively. They are instantiated separately only for values of $\mathbf{C}$ that have actually been received in incoming messages.

Intuitively, protocol AVSS performs a reliable broadcast of $\mathbf{C}$ using the protocol of Bracha [2], where every echo and ready message between two servers P_i and P_j additionally contains the values $f(i,j)$, $f(j,i)$, $f'(i,j)$, and $f'(j,i)$, which they have in common.

The protocol uses $O(n^2)$ messages and has communication complexity $O(\kappa n^4)$. The size of the messages is dominated by $\mathbf{C}$; it can be reduced by a factor of n as shown in Section 3.4.

Protocol AVSS for server P_i and tag $ID.d$ (sharing stage)

```
upon initialization:
    for all $\mathbf{C}$ do
        $e_{\mathbf{C}} \leftarrow 0$; $r_{\mathbf{C}} \leftarrow 0$
        $\mathcal{A}_{\mathbf{C}} \leftarrow \emptyset$; $\mathcal{A}'_{\mathbf{C}} \leftarrow \emptyset$; $\mathcal{B}_{\mathbf{C}} \leftarrow \emptyset$; $\mathcal{B}'_{\mathbf{C}} \leftarrow \emptyset$

upon receiving a message $(ID.d, \texttt{in}, \texttt{share}, s)$:        /* only $P_d$ */
    choose two random bivariate polynomials $f, f' \in \mathbb{Z}_q[x, y]$ of degree $k-1$ with $f(0,0) = f_{00} = s$, i.e.,

        $$f(x,y) = \sum_{j,l=0}^{k-1} f_{jl} x^j y^l \quad \text{and} \quad f'(x,y) = \sum_{j,l=0}^{k-1} f'_{jl} x^j y^l$$

    $\mathbf{C} \leftarrow \{C_{jl}\}$, where $C_{jl} = g^{f_{jl}} h^{f'_{jl}}$ for $j, l \in [0, k-1]$
    for $j \in [1, n]$ do
        $a_j(y) \leftarrow f(j, y)$; $a'_j(y) \leftarrow f'(j, y)$; $b_j(x) \leftarrow f(x, j)$; $b'_j(x) \leftarrow f'(x, j)$
        send the message $(ID.d, \texttt{send}, \mathbf{C}, a_j, a'_j, b_j, b'_j)$ to $P_j$

upon receiving a message $(ID.d, \texttt{send}, \mathbf{C}, a, a', b, b')$ from $P_d$ for the first time:
    if verify-poly($\mathbf{C}, i, a, a', b, b'$) then
        for $j \in [1, n]$ do send the message $(ID.d, \texttt{echo}, \mathbf{C}, a(j), a'(j), b(j), b'(j))$ to $P_j$

upon receiving a message $(ID.d, \texttt{echo}, \mathbf{C}, \alpha, \alpha', \beta, \beta')$ from $P_m$ for the first time:
    if verify-point($\mathbf{C}, i, m, \alpha, \alpha', \beta, \beta'$) then
        $\mathcal{A}_{\mathbf{C}} \leftarrow \mathcal{A}_{\mathbf{C}} \cup \{(m, \alpha)\}$; $\mathcal{A}'_{\mathbf{C}} \leftarrow \mathcal{A}'_{\mathbf{C}} \cup \{(m, \alpha')\}$
        $\mathcal{B}_{\mathbf{C}} \leftarrow \mathcal{B}_{\mathbf{C}} \cup \{(m, \beta)\}$; $\mathcal{B}'_{\mathbf{C}} \leftarrow \mathcal{B}'_{\mathbf{C}} \cup \{(m, \beta')\}$
        $e_{\mathbf{C}} \leftarrow e_{\mathbf{C}} + 1$
        if $e_{\mathbf{C}} = \max\{\lceil \frac{n+t+1}{2} \rceil, k\}$ and $r_{\mathbf{C}} < k$ then
            interpolate $\bar{a}$, $\bar{a}'$, $\bar{b}$, and $\bar{b}'$ from $\mathcal{A}_{\mathbf{C}}$, $\mathcal{A}'_{\mathbf{C}}$, $\mathcal{B}_{\mathbf{C}}$, and $\mathcal{B}'_{\mathbf{C}}$, respectively
            for $j \in [1, n]$ do send the message $(ID.d, \texttt{ready}, \mathbf{C}, \bar{a}(j), \bar{a}'(j), \bar{b}(j), \bar{b}'(j))$ to $P_j$

upon receiving a message $(ID.d, \texttt{ready}, \mathbf{C}, \alpha, \alpha', \beta, \beta')$ from $P_m$ for the first time:
    if verify-point($\mathbf{C}, i, m, \alpha, \alpha', \beta, \beta'$) then
        $\mathcal{A}_{\mathbf{C}} \leftarrow \mathcal{A}_{\mathbf{C}} \cup \{(m, \alpha)\}$; $\mathcal{A}'_{\mathbf{C}} \leftarrow \mathcal{A}'_{\mathbf{C}} \cup \{(m, \alpha')\}$
        $\mathcal{B}_{\mathbf{C}} \leftarrow \mathcal{B}_{\mathbf{C}} \cup \{(m, \beta)\}$; $\mathcal{B}'_{\mathbf{C}} \leftarrow \mathcal{B}'_{\mathbf{C}} \cup \{(m, \beta')\}$
        $r_{\mathbf{C}} \leftarrow r_{\mathbf{C}} + 1$
        if $r_{\mathbf{C}} = k$ and $e_{\mathbf{C}} < \max\{\lceil \frac{n+t+1}{2} \rceil, k\}$ then
            interpolate $\bar{a}$, $\bar{a}'$, $\bar{b}$, and $\bar{b}'$ from $\mathcal{A}_{\mathbf{C}}$, $\mathcal{A}'_{\mathbf{C}}$, $\mathcal{B}_{\mathbf{C}}$, and $\mathcal{B}'_{\mathbf{C}}$, respectively
            for $j \in [1, n]$ do send the message $(ID.d, \texttt{ready}, \mathbf{C}, \bar{a}(j), \bar{a}'(j), \bar{b}(j), \bar{b}'(j))$ to $P_j$
        else if $r_{\mathbf{C}} = k + t$ then
            $\bar{\mathbf{C}} \leftarrow \mathbf{C}$
            $(s_i, s'_i) \leftarrow (\bar{a}(0), \bar{a}'(0))$        /* $(s_i, s'_i)$ is the share of $P_i$ */
            output $(ID.d, \texttt{out}, \texttt{shared})$
```



Protocol AVSS for server P_i and tag $ID.d$ (reconstruction stage)

```
upon receiving a message ($ID.d$, in, reconstruct):
        $c \leftarrow 0; \mathcal{S} \leftarrow \emptyset$
        for $j \in [1, n]$ do send the message ($ID.d$, reconstruct-share, $s_i, s'_i$) to $P_j$

upon receiving a message ($ID.d$, reconstruct-share, $\sigma, \sigma'$) from $P_m$:
        if verify-share($\bar{\mathbf{C}}, m, \sigma, \sigma'$) then
          $\mathcal{S} \leftarrow \mathcal{S} \cup \{(m, \sigma)\}$
          $c \leftarrow c + 1$
          if $c = k$ then
            interpolate $a_0$ from $\mathcal{S}$
            output ($ID.d$, out, reconstructed, $a_0(0)$)
            halt
```

Extended AVSS

Exactly the same as AVSS, except in the ready message, the sender also attaches a signature of the ready message. Collecting $n - t$ such signatures is sufficient to show that the AVSS will run to completion, since at least $n - 2t > k$ number of honest nodes have been dealt valid shares.

NIZKPK of DLEQ (Kate2012)

This generates a NIZKP that s in g^s and $g^s h^{s'}$ are the same.

- Prover:
  - Pick $v_1, v_2 \in_R \mathbb{Z}_p$ and let $t_1 = g^{v_1}$ and $t_2 = h^{v_2}$.
  - Compute hash $c = H(g, h, C_{(g)}(s), C_{(g,h)}(s, r), t_1, t_2)$ where $C_{(g)}(s)$ is a DLog commitment and $C_{(g,h)}(s, r)$ is a Pedersen commitment, and H is a random oracle hash function.
  - Let $u_1 = v_1 - c.\, s$ and $u_2 = v_2 - c.\, r$
  - Send the proof $\pi = (c, u_1, u_2)$ along with $C_{(g)}(s)$ and $C_{(g,h)}(s, r)$
- Verifier:
  - Let $t'_1 = g^{u_1} C_{(g)}(s)^c$ and $t'_2 = h^{u_2} (\frac{C_{(g,h)}(s,r)}{C_{(g)}(s)})^c$
  - Accept proof as valid if $c = H(g, h, C_{(g)}(s), C_{(g,h)}(s, r), t'_1, t'_2)$

Torus DKG

Protocol DKG for tag \(ID\)

- for all $d \in [1, n]$ initialize a sharing of $ID.d$ using extended AVSS
    - node P_d with index d
    - generate $s_d \in_R \mathbb{Z}$
    - for any $\mathbf{C}$, if uninitialized, do
        - $e_{\mathbf{C}} \leftarrow 0, r_{\mathbf{C}} \leftarrow 0, \mathcal{A}_{\mathbf{C}} \leftarrow \emptyset, \mathcal{A'}_{\mathbf{C}} \leftarrow \emptyset$
        - $\mathcal{B}_{\mathbf{C}} \leftarrow \emptyset, \mathcal{B'}_{\mathbf{C}} \leftarrow \emptyset$
    - upon receiving a message $(ID.d, in, share, s_d)$
        - choose two random bivariate polynomials $f, f' \in \mathbb{Z}_q[x, y]$ of degree $k-1$ with $f(0,0) = s_d$, i.e.:
            - $f(x, y) = \sum_{j,l=0}^{k-1} f_{jl} x^j y^l$ and $f'(x, y) = \sum_{j,l=0}^{k-1} f'_{jl} x^j y^l$
        - $\mathbf{C} \leftarrow \{C_{jl}\}$, where $C_{jl} = g^{f_{jl}} h^{f'_{jl}}$ for $j, l \in [0, k-1]$
        - for $j \in [1, n]$ do
            - $a_j(y) \leftarrow f(j, y); a'_j(y) \leftarrow f'(j, y); b_j(x) \leftarrow f(x, j); b'_j \leftarrow f'(x.j)$ send the message $(ID.d, send, \mathbf{C}, a_j, a'_j, b_j, b'_j)$ to P_j
    - upon receiving a message $(ID.d, send, \mathbf{C}, a, a', b, b')$ from P_d for the first time:
        - if $verify\text{-}poly(\mathbf{C}, i, a, a', b, b')$ then
            - for $j \in [1, n]$ do send the message $(ID.d, echo, \mathbf{C}, a(j), a'(j), b(j)b'(j)$ to P_j
    - upon receiving a message $(ID.d, echo, \mathbf{C}, \alpha, \alpha', \beta, \beta')$ from P_m for the first time:
        - if $verify\text{-}point(\mathbf{C}, i, m, \alpha, \alpha', \beta, \beta')$ then
            - $\mathcal{A}_{\mathbf{C}} \leftarrow \mathcal{A}_{\mathbf{C}} \cup \{(m, \alpha)\}; \mathcal{A'}_{\mathbf{C}} \leftarrow \mathcal{A'}_{\mathbf{C}} \cup \{(m, \alpha')\}$
            - $\mathcal{B}_{\mathbf{C}} \leftarrow \mathcal{B}_{\mathbf{C}} \cup \{(m, \beta)\}; \mathcal{B'}_{\mathbf{C}} \leftarrow \mathcal{B'}_{\mathbf{C}} \cup \{(m, \beta')\};$
            - $e_{\mathbf{C}} \leftarrow e_{\mathbf{C}} + 1$
            - if $e_{\mathbf{C}} = max\{\lceil \frac{n+t+1}{2} \rceil, k\}$ and $r_{\mathbf{C}} < k$ then
                - interpolate $\bar{a}, \bar{a}', \bar{b}, \bar{b}''$ from $\mathcal{A}_{\mathbf{C}}, \mathcal{A'}_{\mathbf{C}}, \mathcal{B}_{\mathbf{C}}, \mathcal{B'}_{\mathbf{C}}$
                - for $j \in [1, n]$ do send message $(ID.d, ready, \mathbf{C}, \bar{a}(j), \bar{a}'(j), \bar{b}(j), \bar{b}'(j)$ to P_j
    - upon receiving a message $(ID.d, ready, \mathbf{C}, \alpha, \alpha', \beta, \beta')$ from P_m for the first time:
        - if $verify\text{-}point(\mathbf{C}, i, m, \alpha, \alpha', \beta, \beta')$ then
            - - $\mathcal{A}_{\mathbf{C}} \leftarrow \mathcal{A}_{\mathbf{C}} \cup \{(m, \alpha)\}; \mathcal{A'}_{\mathbf{C}} \leftarrow \mathcal{A'}_{\mathbf{C}} \cup \{(m, \alpha')\}$
            - $\mathcal{B}_{\mathbf{C}} \leftarrow \mathcal{B}_{\mathbf{C}} \cup \{(m, \beta)\}; \mathcal{B'}_{\mathbf{C}} \leftarrow \mathcal{B'}_{\mathbf{C}} \cup \{(m, \beta')\};$
            - $r_{\mathbf{C}} \leftarrow r_{\mathbf{C}} + 1$
            - if $r_{\mathbf{C}} = k$ and $e_{\mathbf{C}} < max\{\lceil \frac{n+t+1}{2} \rceil, k\}$ then

  - interpolate $\bar{a}, \bar{a}', \bar{b}, \bar{b}''$ from $\mathcal{A}_{\mathbf{C}}, \mathcal{A}'_{\mathbf{C}}, \mathcal{B}_{\mathbf{C}}, \mathcal{B}'_{\mathbf{C}}$
  - for $j \in [1, n]$ do send message $(ID.d, ready, \mathbf{C}, \bar{a}(j), \bar{a}'(j), \bar{b}(j), \bar{b}'(j)$ to P_j
- else if $r_{\mathbf{C}} = k + t$ then
  - $\bar{\mathbf{C}} \leftarrow \mathbf{C}$
  - $(s_i, s_i') \leftarrow (\bar{a}(0), \bar{a}'(0))$
  - output $(ID.d, out, shared)$

- wait for $k + t$ sharings on tag $ID.d$ to complete
  - $\mathcal{L}_i \leftarrow \{(d, \Pi_d)\}$ for all sharings $ID.d$
  - propose $\mathcal{L}_i$ in a MVVBA for $ID.propose$ with predicate $verify\text{-}termination$
- wait for MVVBA to decide on some $\mathcal{L}$ for $ID.propose$
  - $\mathcal{J} \leftarrow \{j | (j, \Pi_j) \in \mathcal{L}\}$
- wait for all sharings $ID.j$ for $j \in \mathcal{J}$ to complete
  - for $l \in [0, t]$ do
    - $s_i \leftarrow \sum s_{i_{(ID.j)}}, s_i' \leftarrow \sum s'_{i_{(ID.j)}},$
    - $\bar{D}_l \leftarrow \Pi_{j \in \mathcal{J}} C_{l0}^{[ID.j]}$
    - output $(ID, generated, s_i, s_i', \bar{D})$
- for all $d \in [1, n]$
  - node P_d with index d
  - $\mathcal{N}_{ID} \leftarrow \emptyset, n_{ID} \leftarrow 0$
  - generate $NIZKPK(g^{s_{d_\delta}}, g^{s_{d_\Delta}} h^r, s_{d_\delta} = s_{d_\Delta}) = (d, c, u_1, u_2, g^{s_d}, g^{s_d} h^{s'_d})$
  - for $j \in [1, n]$ do send message $(ID.d, nizkp, \bar{D}, NIZKPK(g^{s_{d_\delta}}, g^{s_{d_\Delta}} h^r, s_{d_\delta} = s_{d_\Delta}))$ to P_j
  - upon receiving a message $(ID.d, nizkp, \bar{D}, NIZKPK(g^{s_{d_\delta}}, g^{s_{d_\Delta}} h^r, s_{d_\delta} = s_{d_\Delta}))$ from P_d for the first time:
    - if $verify\text{-}share\text{-}commitment(g^{s_{d_\Delta}} h^r, \bar{D}, d)$ and $verify\text{-}NIZKPK(c, u_1, u_2, g^{s_d}, g^{s_d} h^{s'_d})$ then
      - $\mathcal{N}_{ID} \leftarrow \mathcal{N}_{ID} \cup (\{d, (c, u_1, u_2, g^{s_d}, g^{s_d} h^{s'_d})\})$
      - $n_{ID} \leftarrow n_{ID} + 1$
    - if $n_{ID} = k$
      - propose $\mathcal{N}_{ID}$ in a MVVBA for $ID.pubkey$ with predicate $verify\text{-}pubkey$
- wait for MVVBA to decide on some $\mathcal{N}_{ID}$ for $ID.pubkey$
  - halt

Verification checks:

- $verify\text{-}poly(\mathbf{C}, i, a, a', b, b')$, where a, a', b, b' are polynomials of degree $k - 1$, i.e.
  - $a(y) = \sum_{l=0}^{k-1} a_l y^l, a'(y) = \sum_{l=0}^{k-1} a_l' y^l, b(x) = \sum_{j=0}^{k-1} b_j x^j, b'(x) = \sum_{j=0}^{k-1} b_j' x^j$

- the predicate verifies that the given polynomials are share and subshare polynomials for P_i consistent with $\mathbf{C}$; it is true if and only if for all $l \in [0, k-1]$, it holds $g^{a_l} h^{a'_l} = \Pi_{j=0}^{k-1} (C_{jl})^{i^j}$ and for all $j \in [0, k-1]$, it holds $g^{b_j} h^{b'_j} = \Pi_{l-0}^{k-1} (C_{jl})^{i^l}$

- *verify-point*$(\mathbf{C}, i, m, \alpha, \alpha', \beta, \beta')$ verifies that the given values $\alpha, \alpha', \beta, \beta'$ correspond to the points $f(m,i), f'(m,i), f(i,m), f'(i,m)$ respectively, commited to in $\mathbf{C}$; it is true if and only if $g^{\alpha} h^{\alpha} = \Pi_{j,l=0}^{k-1} (C_{jl})^{m^j i^l}$ and $g^{\beta} h^{\beta'} = \Pi_{j,l=0}^{k-1} (C_{jl})^{i^j m^l}$.
- *verify-share-commitment*$(g^{s_d} h^r, \bar{D}, d)$ verifies that $g^{s_d} h^r$ forms a valid share of P_d with respect to $\bar{D}$, it is true if and only if $g^{s_d} h^r = \Pi_{j=0}^{k-1} (C_{j0})^{d^j}$
- *verify-termination*$(ID.propose, \mathcal{L})$ verifies that $\mathcal{L}$ contains k tags of AVSS with the proofs that these protocols will actually terminate. It is true if and only if $|\mathcal{L}| = k$ and for ever $(j, \Pi_j) \in \mathcal{L}$ the list Π_j contains at least $k+t$ valid signatures on the message $(ID.j, ready)$ from distinct nodes.
- *verify-NIZKP*$(c, u_1, u_2, g^{s_d}, g^{s_d} h^{s'_d})$ verifies that for $s_{d_\delta} \leftarrow s_d | g^{s_d}$, $s_{d_\Delta} \leftarrow s_d | g^{s_d} h^{s'_d}$, $s_{d_\delta} = s_{d_\Delta}$. It is true if and only if $c = H(g, h, C_{(g)}(s), C_{(g,h)}(s,r), t'_1, t'_2$ for $t'_1 = g^{u_1} C_{(g)}(s)^c$, $t'_2 = h^{u_2} (\frac{C_{(g,h)}(s,r)}{C_{(g)}(s)})^c$
- *verify-pubkey*$(\mathcal{N}_{ID}, \bar{D})$ verifies that for $s \in \mathcal{S} \leftarrow g^{s_d} h^{s'_d} | \{d, (c, u_1, u_2, g^{s_d}, g^{s_d} h^{s'_d})\} \in \mathcal{N}_{ID}$, $s = \Pi_{j=0}^{k-1} (\bar{D}_{j0})^{d^j}$ and that for $N \in \mathcal{N}_{ID}$, *verify-NIZKP*(N) is true.

Liveness

We have to show that all wait states that a node enters are eventually satisfied.

A node waiting on k AVSS sharings of tag $ID.d$ to complete will receive them since there are at least k honest nodes who have started AVSS, and AVSS satisfies the liveness property as proven by Cachin2002 so it will complete.

A node waiting on MVVBA to decide on some $\mathcal{L}$, by the definition of MVVBA, subject to the constraints on Tendermint, will also receive a decision.

A node waiting on all sharings $ID.j, j \in \mathcal{J}$, will receive them since by the definition of *verify-termination*, there are at least $k+t$ valid signatures on the *ready* message, thus at least k honest servers have sent a *ready* message for each sharing $ID.j$, which is sufficient to guarantee termination of the sharing.

Similarly, a node waiting on MVVBA to decide on $\mathcal{N}_{ID}$, by the definition of MVVBA, subject to the constraints on Tendermint, will also receive a decision, and halt.

Correctness

By the properties of MVVBA, all honest nodes decide on the same value of selected sets of AVSS $ID.d, d \in \mathcal{J}$, so when an honest node has completed the DKG protocol, they have $|\mathcal{J}|$ subshares such that $s_i \leftarrow \sum s_{i_{(ID.j)}}, s'_i \leftarrow \sum s'_{i_{(ID.j)}}, j \in \mathcal{J}$.

Since each AVSS satisfies the correctness property, the DKG is a known linear combination of AVSS subsharings on $ID.j \in \mathcal{J}$, and Lagrange interpolation is homomorphic over addition, the DKG also satisfies the correctness property.

Secrecy

Since $s_0 = \sum_{d \in \mathcal{J}} s_d$, and $|\mathcal{J}| = k$, and $k > t$, at least one instance of AVSS is by an honest node. Wlog, assume $k = t + 1$, so there is only a single AVSS instance which is by an honest node. For some given $s_M = s_0 - s_H$, where s_H is the secret for the AVSS for the honest node, since s_H is uniformly random, s_0 is also uniformly random to the adversary.
Assume an adversary algorithm that can compute s_0 given g, D and t degree-t polynomials consistent with D. Then given a DLog instance g, g^α, it is always possible to compute some $\tilde{D}$ with $\tilde{D}_{00} = g^\alpha$ by generating t degree-t polynomials, and interpolating (in the exponent) the other commitments in the commitment matrix. We can then present $\tilde{D}$ along with the constant terms of each of the t degree-t polynomials to solve for α, thus breaking discrete logarithm.

Efficiency

Since we have n runs of AVSS (which has uniformly bounded communication complexity), and two MVVBA subprotocols on Tendermint (which also has uniformly bounded communication complexity), the overall DKG has uniformly bounded communication complexity.