ADAPTIVITY AND ASYNCHRONY IN DISTRIBUTED KEY GENERATION

SARAH MEIKLEJOHN (GOOGLE & UCL)
A distributed key generation (DKG) protocol allows a set of participants to generate a threshold public key.
USING A DKG

Classical use cases:
- Want $t$ of $n$ parties to have to collaborate to **decrypt** something
- Want $t$ of $n$ parties to have to collaborate to authorize some action (**sign** something)

For these we expect to run the DKG **only once**

 VERIFY($pk, \sigma, m) = 1$
USING A DKG

Classical use cases:
- Want t of n parties to have to collaborate to decrypt something
- Want t of n parties to have to collaborate to authorize some action (sign something)

For these we expect to run the DKG only once

Can also use DKGs for random beacons
1. Run the DKG to generate a threshold public key
2. Have t parties produce a unique threshold signature
3. Hash the unique signature to produce randomness

Here we might run the DKG many times, so there is interest in having efficient DKG protocols that operate in asynchronous environments
<table>
<thead>
<tr>
<th></th>
<th>word complexity</th>
<th>round complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>[KG10]*</td>
<td>$n^4$</td>
<td>$n$</td>
</tr>
<tr>
<td>[KMS20]</td>
<td>$n^3$</td>
<td>$n$</td>
</tr>
<tr>
<td>[APMMST21]</td>
<td>$n^3$</td>
<td>1</td>
</tr>
<tr>
<td>[DYXMKR22]</td>
<td>$n^3$</td>
<td>$\log(n)$</td>
</tr>
<tr>
<td>[GS22]</td>
<td>$n^3$</td>
<td>1</td>
</tr>
</tbody>
</table>

*assumes partial synchrony
BUILDING A DKG

Most DKGs are based on **secret sharing**

A secret sharing scheme consists of two protocols:
- **Deal (Share)** allows one party (the **dealer**) to share a secret
- **Reconstruct** allows t+1 parties to compute the secret
Building a DKG

Most DKGs are based on secret sharing

A secret sharing scheme consists of two protocols:
- Deal (Share) allows one party (the dealer) to share a secret
- Reconstruct allows t+1 parties to compute the secret

Shamir secret sharing of a secret s:
Deal:
- Form a random degree-t polynomial p(x) such that p(0) = s
- Send p(i) to party i
Reconstruct:
- Party i shares p(i) with other parties
- Once t+1 parties have shared points, can reconstruct p(x) using Lagrange interpolation
A typical DKG

have everyone act as the dealer in parallel

Deal

$p_6$ (5)

$p_5$

$p_4$ (4)

$p_3$ (3)

$p_2$ (2)

$p_1$ (1)
A TYPICAL DKG

- party i shares $g^{p_j(i)}$ for all j
- each party interpolates (in the exponent) to get $g^{p_j}$ and computes $\prod g^{p_j} = g^{\sum p_j} = g^p$
- each party evaluates (in the exponent) to compute and output $pk = g^{p(0)}$

perform reconstruction in the exponent
WHEN THINGS GO WRONG

*p5* → *Deal* → *p6(5) → p4*

*p6* → *Deal* → *p6(1) → p1*

*p6* → *Deal* → *p6(2) → p2*

*p6* → *Deal* → *p6(3) → p3*
WHEN THINGS GO WRONG

VSS.Deal

p_6

\[ p_5 \quad p_6(5) \]

\[ p_4 \]

\[ p_1 \]

\[ p_6(1) \]

\[ p_2 \]

\[ p_6(2) \]

\[ p_3 \]

\[ p_6(3) \]

this is why typical DKGs use verifiable secret sharing...

...and need an extra complaints round to agree on which dealers’ polynomials to include
BUILDING A DKG

1. Party i:
   - acts as the VSS dealer
   - participates in VSS sharing for all other parties j
2. All parties agree on a set of dealers D using a complaints round
3. Party i reconstructs, in the exponent, the sum of secrets for dealers in D

so the best way to get a better (A)DKG is to build a better (A)VSS
BINGO [AJMMS23]

Bingo is an AVSS that:
- allows secrets to be packed (share f+1 secrets with the same complexity as one)
- has optimal resilience (n = 3f + 1)
- has $O(n^2)$ word complexity and $O(1)$ round complexity
- allows for adaptive corruptions (new definitions of VSS termination, correctness, and secrecy)
SHARING IN BINGO

- sample $\phi(X, Y)$ s.t. $\phi(-k, 0) = s_k$ for all secrets $s_k$ (packing)
- broadcast commitment* to $\phi(X, Y)$
- set $\alpha_i = \phi(X, i)$, $\beta_i = \phi(i, Y)$ (meaning $\alpha_i(j) = \beta_j(i)$)
- send $\alpha_i$ to party $i$

the goal in Share is for each party $i$ to learn their $\alpha_i$ polynomial

*using a natural extension of KZG to bivariate polynomials
the goal in Share is for each party $i$ to learn their $\alpha_i$ polynomial
### SHARING IN BINGO

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha_1 )</td>
<td>( \beta_1 ) ( \beta_2 ) ( \beta_3 ) ( \beta_4 ) ( \beta_5 )</td>
</tr>
<tr>
<td>( \alpha_2 )</td>
<td>( \beta_{1,1} ) ( \beta_{1,2} ) ( \beta_{1,3} ) ( \beta_{1,4} ) ( \beta_{1,5} )</td>
</tr>
<tr>
<td>( \alpha_3 )</td>
<td>( \beta_{2,1} ) ( \beta_{2,2} ) ( \beta_{2,3} ) ( \beta_{2,4} ) ( \beta_{2,5} )</td>
</tr>
<tr>
<td>( \alpha_4 )</td>
<td>( \beta_{3,1} ) ( \beta_{3,2} ) ( \beta_{3,3} ) ( \beta_{3,4} ) ( \beta_{3,5} )</td>
</tr>
<tr>
<td>( \alpha_5 )</td>
<td>( \beta_{4,1} ) ( \beta_{4,2} ) ( \beta_{4,3} ) ( \beta_{4,4} ) ( \beta_{4,5} )</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>( \beta_{5,1} ) ( \beta_{5,2} ) ( \beta_{5,3} ) ( \beta_{5,4} ) ( \beta_{5,5} )</td>
</tr>
</tbody>
</table>
SHARING IN BINGO

\[ \begin{align*}
\alpha_1 & \rightarrow \beta_1, \beta_2, \beta_3, \beta_4, \beta_5 \\
\alpha_2 & \rightarrow v_{1,1}, v_{1,2}, v_{1,3}, v_{1,4}, v_{1,5} \\
\alpha_3 & \rightarrow v_{2,1}, v_{2,2}, v_{2,3}, v_{2,4}, v_{2,5} \\
\alpha_4 & \rightarrow v_{3,1}, v_{3,2}, v_{3,3}, v_{3,4}, v_{3,5} \\
\alpha_5 & \rightarrow v_{4,1}, v_{4,2}, v_{4,3}, v_{4,4}, v_{4,5} \\
\alpha_6 & \rightarrow v_{5,1}, v_{5,2}, v_{5,3}, v_{5,4}, v_{5,5}
\end{align*} \]
send $\alpha_5(j)$ to party $j$ 
$\Rightarrow$ party $j$ learns $\beta_j(5)$
## Sharing in Bingo

Given enough points, party 1 can interpolate to learn $\beta_1$.
SHARING IN BINGO

send $\beta_1(j)$ to party $j$

$\Rightarrow$ party $j$ learns $\alpha_j(1)$
given enough points, party 2 can interpolate to learn $\alpha_2$!
Some hidden complexities:
- Parties have to prove correctness of their evaluations (more work for the commitment)
- All parties need to have the same commitment (use reliable broadcast [DXR21])
- Adaptivity!
- sample $\phi(X, Y)$ s.t. $\phi(-k, 0) = s_k$ for all secrets $s_k$ (packing)
  - party $j$ shares $\alpha_j(-k)$
  - interpolates $\beta_{-k}$
  - evaluates $s_k = \beta_{-k}(0)$

can reconstruct one secret at a time

same old trick ($\alpha_j(-k) = \beta_{-k}(j)$)
**RECONSTRUCTION IN BINGO**

- party $j$ gets $\alpha_{j,i}$ when $i$ is dealing
- computes $\alpha_j = \Sigma \alpha_{j,i}$
- shares $\alpha_j(-k)$
- interpolates $\beta_{-k}$
- evaluates $s_k = \beta_{-k}(0)$

(can also reconstruct sums of secrets with the same complexity!)

(same for the batch reconstruction of multiple secrets)
1. Party i:
   - acts as the VSS dealer
   - participates in VSS sharing for all other parties j
2. All parties agree on a set of dealers D using a complaints round
3. Party i reconstructs, in the exponent, the sum of secrets for dealers in D
1. Party i:
   - acts as the Bingo dealer
   - participates in Bingo sharing for all other parties j
2. All parties agree on a set of dealers D using a complaints round
3. Party i reconstructs, in the exponent, the sum of secrets for dealers in D
   - computes $\alpha_j = \sum \alpha_{j,i}$ for j in D
   - shares $g^{\alpha_j(-k)}$
   - interpolates and evaluates to output $pk = g^{\beta-k(0)}$

---

can do this by sending one point rather than $O(n)$
BUILDING A DKG

1. Party i:
   - acts as the Bingo dealer
   - participates in Bingo sharing for all other parties j

2. All parties agree on a set of dealers D using a complaints round

3. Party i reconstructs, in the exponent, the sum of secrets for dealers in D
   - computes $\alpha_j = \Sigma \alpha_{j,i}$ for j in D
   - shares $g^{\alpha_j(-k)}$
   - interpolates and evaluates to output $pk = g^{\beta-k(0)}$
VABA

Validated asynchronous Byzantine agreement (VABA) allows parties to agree on a valid value
- all non-faulty parties complete the protocol and output the same value
- this value is valid according to some external validity function checkValidity

For us, checkValidity(dealers, sigs) outputs 1 iff:
- \(|\text{dealers}| \geq f+1\)
- \(|\text{sigs}| \geq f+1\)
- Verify(pk_j, \sigma_j, \text{dealers}) for all \((j, \sigma_j)\) in sigs
any set that passes verification must be a super-set of this common core

built this based on reliable broadcast
VABA [AJMMST21]

built this based on threshold VRFs and verifiable gather
VABA [AJMMSST21]

validates asynchronous Byzantine agreement (VABA)

built this (“No Waitin’ Hotstuff”) based on proposal election
ADAPTIVELY SECURE VABA

Based on Bingo

- Verifiable gather
- Adaptive proposal election
- Adaptively secure validated asynchronous Byzantine agreement (VABA)
## ADKG Protocols

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[KG10]*</td>
<td>$n^4$</td>
<td>$n$</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>[KMS20]</td>
<td>$n^3$</td>
<td>$n$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>[APM MST21]</td>
<td>$n^3$</td>
<td>1</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>[DYXMKR22]</td>
<td>$n^3$</td>
<td>$\log(n)$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>[GS22]</td>
<td>$n^3$</td>
<td>1</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Our work</td>
<td>$n^3$</td>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

*assumes partial synchrony*
THANKS!
ANY QUESTIONS?