Attacks and Fixes on Distributed Key Generation Protocols

Chelsea Komlo, University of Waterloo

Naval Postgraduate School, November 23, 2021

Distributed Key Generation (informal)

- Generating key material without relying on a trusted entity is often desirable for distributed protocols.
- In a DKG, n parties (chosen beforehand) participate, all equally trusted.
- Goal is to:
  - Generate a secret that all parties contribute to but no party knows
  - t parties are required to recover.
  - While t can equal n, the setting where $t \leq n$ is often desirable.

Distributed Key Generation (informal)

- Generating key material without relying on a trusted entity is often desirable for distributed protocols.
- In a DKG, n parties (chosen beforehand) participate, all equally trusted.
- Goal is to:
  - Generate a secret that all parties contribute to but no party knows
  - t parties are required to recover.
  - While t can equal n, the setting where $t \leq n$ is often desirable.

Distributed Key Generation (informal)

- Generating key material without relying on a trusted entity is often desirable for distributed protocols.
- In a DKG, n parties (chosen beforehand) participate, all equally trusted.
- Goal is to:
  - Generate a secret that all parties contribute to but no party knows
  - t parties are required to recover.
  - While t can equal n, the setting where $t \leq n$ is often desirable.

Distributed Key Generation (informal)

- Generating key material without relying on a trusted entity is often desirable for distributed protocols.
- In a DKG, n parties (chosen beforehand) participate, all equally trusted.
- Goal is to:
  - Generate a secret that all parties contribute to but no party knows
  - t parties are required to recover.
  - While t can equal n, the setting where $t \leq n$ is often desirable.

Distributed Key Generation (informal)

- Generating key material without relying on a trusted entity is often desirable for distributed protocols.
- In a DKG, n parties (chosen beforehand) participate, all equally trusted.
- Goal is to:
  - Generate a secret that all parties contribute to but no party knows
  - t parties are required to recover.
  - While t can equal n, the setting where $t \leq n$ is often desirable.

Distributed Key Generation (informal)

- Generating key material without relying on a trusted entity is often desirable for distributed protocols.
- In a DKG, n parties (chosen beforehand) participate, all equally trusted.
- Goal is to:
  - Generate a secret that all parties contribute to but no party knows
  - t parties are required to recover.
  - While t can equal n, the setting where $t \leq n$ is often desirable.

DKG Threat Model

- At minimum, requires assuming the adversary can control up to $t-1$ players.
- If the adversary controlled t players, they could trivially recover the secret.

DKG Threat Model

- At minimum, requires assuming the adversary can control up to $t-1$ players.
- If the adversary controlled t players, they could trivially recover the secret.

Use Cases for DKGs

- Key generation for threshold or multiparty signatures
- Distributed PRFs (League of Entropy)
- Key generation for anonymous token issuance (Privacy Pass)

Use Cases for DKGs

- Key generation for threshold or multiparty signatures
- Distributed PRFs (League of Entropy)
- Key generation for anonymous token issuance (Privacy Pass)

Use Cases for DKGs

- Key generation for threshold or multiparty signatures
- Distributed PRFs (League of Entropy)
- Key generation for anonymous token issuance (Privacy Pass)

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow\$ \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.
- After having received all $c_1, \ldots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow\$ \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.
- After having received all $c_1, \ldots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow\$ \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.
- After having received all $c_1, \ldots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow\$ \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.
- After having received all $c_1, \dots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow_{\$} \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.
- After having received all $c_1, \ldots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow\!\!\$\, \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.
- After having received all $c_1, \ldots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow\$ \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.
- After having received all $c_1, \ldots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow\!\!\$\, \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.
- After having received all $c_1, \ldots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow\$ \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.
- After having received all $c_1, \ldots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

A Trivial n-out-of-n DKG

- Each party i begins by:
  - Sampling $x_i \leftarrow\!\$\, \mathbb{Z}_q$
  - Generating $X_i \leftarrow g^x$
  - Generating the commitment $c_i \leftarrow H(X_i)$
  - Each party publishes their commitment to all other parties.

- After having received all $c_1, \ldots, c_n$, each party publishes X_i
- Each party checks that $c_i \stackrel{?}{=} H(X_i)$. If not, they abort and identify misbehaving parties.
- Otherwise:
  - $pk \leftarrow \prod X_i$
  - While unknown to any party, $sk \leftarrow \sum x_i$

t-out-of-*n* DKGs

- In practice, it is desirable to require only a *subset* of the parties to be online.
- For example, secret keys may be lost or a server might be unavailable.
- t is referred to as the *threshold* number of participants required to participate.
- n is the total number of participants.

t-out-of-n DKGs

- In practice, it is desirable to require only a *subset* of the parties to be online.
- For example, secret keys may be lost or a server might be unavailable.
- t is referred to as the *threshold* number of participants required to participate.
- n is the total number of participants.

t-out-of-n DKGs

- In practice, it is desirable to require only a *subset* of the parties to be online.
- For example, secret keys may be lost or a server might be unavailable.
- t is referred to as the *threshold* number of participants required to participate.
- n is the total number of participants.

t-out-of-n DKGs

- In practice, it is desirable to require only a *subset* of the parties to be online.
- For example, secret keys may be lost or a server might be unavailable.
- t is referred to as the *threshold* number of participants required to participate.
- n is the total number of participants.

Background

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among *n* participants, where *t* participants must cooperate to recover α.
- *f* is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that *t* points uniquely define a polynomial of degree *t* – 1!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Verifiable Secret Sharing

- Allows participants to verify their share $w_i = f(i)$ is on the same polynomial as all other participants, *without* revealing f directly.

- Working in the discrete log setting: a commitment to f is $\vec{D} \leftarrow \langle A_0, A_1, \ldots, A_{t-1} \rangle$, where

$$A_0 \leftarrow g^{\alpha},\ A_1 \leftarrow g^{a_1}, \ldots$$

- Verification of shares requires performing polynomial interpolation in the exponent to check that $g^{f(i)}$ is a point on g^f.

Verifiable Secret Sharing

- Allows participants to verify their share $w_i = f(i)$ is on the same polynomial as all other participants, *without* revealing f directly.
- Working in the discrete log setting: a commitment to f is $\vec{D} \leftarrow \langle A_0, A_1, \ldots, A_{t-1} \rangle$, where

$$A_0 \leftarrow g^{\alpha},\ A_1 \leftarrow g^{a_1}, \ldots$$

- Verification of shares requires performing polynomial interpolation in the exponent to check that $g^{f(i)}$ is a point on g^f.

Verifiable Secret Sharing

- Allows participants to verify their share $w_i = f(i)$ is on the same polynomial as all other participants, *without* revealing f directly.

- Working in the discrete log setting: a commitment to f is $\vec{D} \leftarrow \langle A_0, A_1, \ldots, A_{t-1} \rangle$, where

$$A_0 \leftarrow g^{\alpha},\ A_1 \leftarrow g^{a_1}, \ldots$$

- Verification of shares requires performing polynomial interpolation in the exponent to check that $g^{f(i)}$ is a point on g^f.

Rushing Adversary

- In a multi-party protocol where scheduling of communication is not enforced, an adversary can choose when to participate.
- A *rushing adversary* is one that waits to “speak last”, and so learns every other participant’s contributions before selecting their own.

Rushing Adversary

- In a multi-party protocol where scheduling of communication is not enforced, an adversary can choose when to participate.
- A *rushing adversary* is one that waits to “speak last”, and so learns every other participant’s contributions before selecting their own.

Distributed Key Generation

DKG Definition

A DKG is the tuple (KeyGen, Recover), such that:

- $KeyGen(\lambda, n, t) \rightarrow (pk, \text{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $Recover(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

DKG Definition

A DKG is the tuple (KeyGen, Recover), such that:

- $KeyGen(\lambda, n, t) \rightarrow (pk, \text{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $Recover(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

DKG Definition

A DKG is the tuple (KeyGen, Recover), such that:

- $KeyGen(\lambda, n, t) \rightarrow (pk, \text{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $Recover(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

DKG Definition

A DKG is the tuple $(\text{KeyGen}, \text{Recover})$, such that:

- $KeyGen(\lambda, n, t) \rightarrow (pk, \text{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $Recover(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

DKG Definition

A DKG is the tuple $(\text{KeyGen}, \text{Recover})$, such that:

- $KeyGen(\lambda, n, t) \rightarrow (pk, \text{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $Recover(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

DKG Definition

A DKG is the tuple (KeyGen, Recover), such that:

- $KeyGen(\lambda, n, t) \rightarrow (pk, \text{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $Recover(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

DKG Definition

A DKG is the tuple (KeyGen, Recover), such that:

- $KeyGen(\lambda, n, t) \rightarrow (pk, \mathrm{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $Recover(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

DKG Definition

A DKG is the tuple (KeyGen, Recover), such that:

- $\mathit{KeyGen}(\lambda, n, t) \rightarrow (pk, \text{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $\mathit{Recover}(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

DKG Definition

A DKG is the tuple $(\text{KeyGen}, \text{Recover})$, such that:

- $KeyGen(\lambda, n, t) \rightarrow (pk, \text{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $Recover(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

DKG Definition

A DKG is the tuple (KeyGen, Recover), such that:

- $KeyGen(\lambda, n, t) \rightarrow (pk, \text{qual}, \{sk_1, \ldots, sk_n\})$
  - A probabilistic protocol among n predetermined parties.
  - Output to each party includes:
    1. Public key pk
    2. The set qual of parties remaining at the end.
    3. Their secret key share sk_i.
- $Recover(\{sk_i\}_C) \rightarrow sk$
  - A deterministic algorithm performed by one entity.
  - Assuming $|C| \geq t$, sk is recovered by combining $\{sk_i\}_C$.

On Determining qual

- Because cheating parties can be picked out during protocol execution,

$$t \leq |\text{qual}| \leq n$$

- If qual $\ngeq t$, then the DKG simply fails, as t parties are required for *Recover*.
- Parties perform a sub-protocol to identify and kick out cheaters.

On Determining qual

- Because cheating parties can be picked out during protocol execution,

$$t \leq |\text{qual}| \leq n$$

- If qual $\ngeq t$, then the DKG simply fails, as t parties are required for *Recover*.

- Parties perform a sub-protocol to identify and kick out cheaters.

On Determining qual

- Because cheating parties can be picked out during protocol execution,

$$t \leq |\mathrm{qual}| \leq n$$

- If qual $\not\geq t$, then the DKG simply fails, as t parties are required for *Recover*.

- Parties perform a sub-protocol to identify and kick out cheaters.

Correctness of a DKG

- All subsets of t shares define the same secret key sk (or any subset fulfilling the required access structure)
- All parties that honestly followed the protocol have the same value of the public key pk.

Gennaro, Rosario and Jarecki, Stanislaw and Krawczyk, Hugo and Rabin, Tal. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. Journal of Cryptology, 2007

Correctness of a DKG

- All subsets of t shares define the same secret key sk (or any subset fulfilling the required access structure)
- All parties that honestly followed the protocol have the same value of the public key pk.

Gennaro, Rosario and Jarecki, Stanislaw and Krawczyk, Hugo and Rabin, Tal. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. Journal of Cryptology, 2007

Security: Two Approaches

- *Stand-Alone*: The DKG can be proven secure without reference to how it is used.
  - Nothing about *sk* is revealed beyond what is revealed by *pk* (from GJKR).
  - The protocol can be perfectly simulated to an adversary for a challenge public key.
  - In other words, the simulated DKG must output the challenge as the group's public key.
- *Contextual*: Prove the security of the DKG in the context of demonstrating security of the protocol in which it is used [2]

[2] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, Alin Tomescu. Aggregatable Distributed Key Generation, EUROCRYPT 2021

Security: Two Approaches

- *Stand-Alone*: The DKG can be proven secure without reference to how it is used.
    - Nothing about *sk* is revealed beyond what is revealed by *pk* (from GJKR).
    - The protocol can be perfectly simulated to an adversary for a challenge public key.
    - In other words, the simulated DKG must output the challenge as the group's public key.
- *Contextual*: Prove the security of the DKG in the context of demonstrating security of the protocol in which it is used [2]

[2] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, Alin Tomescu. Aggregatable Distributed Key Generation, EUROCRYPT 2021

Security: Two Approaches

- *Stand-Alone*: The DKG can be proven secure without reference to how it is used.
  - Nothing about *sk* is revealed beyond what is revealed by *pk* (from GJKR).
  - The protocol can be perfectly simulated to an adversary for a challenge public key.
  - In other words, the simulated DKG must output the challenge as the group's public key.
- *Contextual*: Prove the security of the DKG in the context of demonstrating security of the protocol in which it is used [2]

[2] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, Alin Tomescu. Aggregatable Distributed Key Generation, EUROCRYPT 2021

Security: Two Approaches

- *Stand-Alone*: The DKG can be proven secure without reference to how it is used.
  - Nothing about *sk* is revealed beyond what is revealed by *pk* (from GJKR).
  - The protocol can be perfectly simulated to an adversary for a challenge public key.
  - In other words, the simulated DKG must output the challenge as the group's public key.

- *Contextual*: Prove the security of the DKG in the context of demonstrating security of the protocol in which it is used [2]

[2] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, Alin Tomescu. Aggregatable Distributed Key Generation, EUROCRYPT 2021

Security: Two Approaches

- *Stand-Alone*: The DKG can be proven secure without reference to how it is used.
  - Nothing about *sk* is revealed beyond what is revealed by *pk* (from GJKR).
  - The protocol can be perfectly simulated to an adversary for a challenge public key.
  - In other words, the simulated DKG must output the challenge as the group's public key.
- *Contextual*: Prove the security of the DKG in the context of demonstrating security of the protocol in which it is used [2]

[2] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, Alin Tomescu. Aggregatable Distributed Key Generation, EUROCRYPT 2021

Security: Two Approaches (cont'd)

- *Stand-Alone*:
  - Pro: Proving security means that the DKG can be used in any context.
  - Con: Very hard definition to achieve, requires guaranteed output delivery.
    - No notion of failure and/or rewinding in the existing definition.
- *Contextual*:
  - Pro: Allows for more efficient protocols to be proven secure (more on this later).
  - Con: Proofs are unwieldy and requires proving the security of the DKG over and over in different use cases.

Security: Two Approaches (cont'd)

- *Stand-Alone*:
  - Pro: Proving security means that the DKG can be used in any context.
  - Con: Very hard definition to achieve, requires guaranteed output delivery.
    - No notion of failure and/or rewinding in the existing definition.
- *Contextual*:
  - Pro: Allows for more efficient protocols to be proven secure (more on this later).
  - Con: Proofs are unwieldy and requires proving the security of the DKG over and over in different use cases.

Security: Two Approaches (cont'd)

- *Stand-Alone*:
  - Pro: Proving security means that the DKG can be used in any context.
  - Con: Very hard definition to achieve, requires guaranteed output delivery.
    - No notion of failure and/or rewinding in the existing definition.
- *Contextual*:
  - Pro: Allows for more efficient protocols to be proven secure (more on this later).
  - Con: Proofs are unwieldy and requires proving the security of the DKG over and over in different use cases.

Security: Two Approaches (cont'd)

- *Stand-Alone*:
  - Pro: Proving security means that the DKG can be used in any context.
  - Con: Very hard definition to achieve, requires guaranteed output delivery.
    - No notion of failure and/or rewinding in the existing definition.
- *Contextual*:
  - Pro: Allows for more efficient protocols to be proven secure (more on this later).
  - Con: Proofs are unwieldy and requires proving the security of the DKG over and over in different use cases.

Security: Two Approaches (cont'd)

- *Stand-Alone*:
  - Pro: Proving security means that the DKG can be used in any context.
  - Con: Very hard definition to achieve, requires guaranteed output delivery.
    - No notion of failure and/or rewinding in the existing definition.
- *Contextual*:
  - Pro: Allows for more efficient protocols to be proven secure (more on this later).
  - Con: Proofs are unwieldy and requires proving the security of the DKG over and over in different use cases.

Security: Two Approaches (cont'd)

- *Stand-Alone*:
  - Pro: Proving security means that the DKG can be used in any context.
  - Con: Very hard definition to achieve, requires guaranteed output delivery.
    - No notion of failure and/or rewinding in the existing definition.
- *Contextual*:
  - Pro: Allows for more efficient protocols to be proven secure (more on this later).
  - Con: Proofs are unwieldy and requires proving the security of the DKG over and over in different use cases.

Security: Two Approaches (cont'd)

- *Stand-Alone*:
  - Pro: Proving security means that the DKG can be used in any context.
  - Con: Very hard definition to achieve, requires guaranteed output delivery.
    - No notion of failure and/or rewinding in the existing definition.
- *Contextual*:
  - Pro: Allows for more efficient protocols to be proven secure (more on this later).
  - Con: Proofs are unwieldy and requires proving the security of the DKG over and over in different use cases.

Requirement of Security: Uniform Distribution

- In a single-party setting, ensuring that a secret key is randomly sampled is easy.
- In a multi-party setting, where the adversary is a participant, ensuring key material is uniformly distributed is harder.
- For example: $\gamma \leftarrow \alpha + \sum_{i=1}^{n} \beta_i$
  - Here, $\alpha \leftarrow_{\$} \mathbb{Z}_q$ and each b_i is chosen non-uniformly.
  - γ is random if each $\beta_i \in \mathbb{Z}_q$ is chosen *without* knowledge of α.
  - Otherwise, γ is not random.

Requirement of Security: Uniform Distribution

- In a single-party setting, ensuring that a secret key is randomly sampled is easy.

- In a multi-party setting, where the adversary is a participant, ensuring key material is uniformly distributed is harder.

- For example: $\gamma \leftarrow \alpha + \sum_{i=1}^{n} \beta_i$
  - Here, $\alpha \leftarrow_{\$} \mathbb{Z}_q$ and each b_i is chosen non-uniformly.
  - γ is random if each $\beta_i \in \mathbb{Z}_q$ is chosen *without* knowledge of α.
  - Otherwise, γ is not random.

Requirement of Security: Uniform Distribution

- In a single-party setting, ensuring that a secret key is randomly sampled is easy.

- In a multi-party setting, where the adversary is a participant, ensuring key material is uniformly distributed is harder.

- For example: $\gamma \leftarrow \alpha + \sum_{i=1}^{n} \beta_i$
  - Here, $\alpha \leftarrow\!\!\$\, \mathbb{Z}_q$ and each b_i is chosen non-uniformly.
  - γ is random if each $\beta_i \in \mathbb{Z}_q$ is chosen *without* knowledge of α.
  - Otherwise, γ is not random.

Requirement of Security: Uniform Distribution

- In a single-party setting, ensuring that a secret key is randomly sampled is easy.
- In a multi-party setting, where the adversary is a participant, ensuring key material is uniformly distributed is harder.
- For example: $\gamma \leftarrow \alpha + \sum_{i=1}^{n} \beta_i$
  - Here, $\alpha \leftarrow_{\$} \mathbb{Z}_q$ and each b_i is chosen non-uniformly.
  - γ is random if each $\beta_i \in \mathbb{Z}_q$ is chosen *without* knowledge of α.
  - Otherwise, γ is not random.

Requirement of Security: Uniform Distribution

- In a single-party setting, ensuring that a secret key is randomly sampled is easy.

- In a multi-party setting, where the adversary is a participant, ensuring key material is uniformly distributed is harder.

- For example: $\gamma \leftarrow \alpha + \sum_{i=1}^{n} \beta_i$
  - Here, $\alpha \leftarrow_{\$} \mathbb{Z}_q$ and each b_i is chosen non-uniformly.
  - γ is random if each $\beta_i \in \mathbb{Z}_q$ is chosen *without* knowledge of α.
  - Otherwise, γ is not random.

Requirement of Security: Uniform Distribution

- In a single-party setting, ensuring that a secret key is randomly sampled is easy.

- In a multi-party setting, where the adversary is a participant, ensuring key material is uniformly distributed is harder.

- For example: $\gamma \leftarrow \alpha + \sum_{i=1}^{n} \beta_i$
  - Here, $\alpha \leftarrow\!\$\, \mathbb{Z}_q$ and each b_i is chosen non-uniformly.
  - γ is random if each $\beta_i \in \mathbb{Z}_q$ is chosen *without* knowledge of α.
  - Otherwise, γ is not random.

Current Landscape

Pedersen DKG: Overview

- Each participant *i* acts as the dealer and performs a Shamir secret sharing of a secret α_i, distributing shares w_{ij} to each other.
- The group's secret at the end is $sk \leftarrow \sum \alpha_i = \sum \sum w_{ij}\lambda_i$.
- Proven secure in the context of threshold signatures. [3]
- A rushing adversary *can* bias key material, but by a limited amount.

[3] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, Tal Rabin. Secure Applications of Pedersen's Distributed Key Generation Protocol, CT-RSA, 2003.

Pedersen DKG: Overview

- Each participant i acts as the dealer and performs a Shamir secret sharing of a secret α_i, distributing shares w_{ij} to each other.
- The group's secret at the end is $sk \leftarrow \sum \alpha_i = \sum \sum w_{ij} \lambda_i$.
- Proven secure in the context of threshold signatures. [3]
- A rushing adversary *can* bias key material, but by a limited amount.

[3] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, Tal Rabin. Secure Applications of Pedersen's Distributed Key Generation Protocol, CT-RSA, 2003.

Pedersen DKG: Overview

- Each participant i acts as the dealer and performs a Shamir secret sharing of a secret α_i, distributing shares w_{ij} to each other.
- The group's secret at the end is $sk \leftarrow \sum \alpha_i = \sum \sum w_{ij} \lambda_i$.
- Proven secure in the context of threshold signatures. [3]
- A rushing adversary *can* bias key material, but by a limited amount.

[3] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, Tal Rabin. Secure Applications of Pedersen's Distributed Key Generation Protocol, CT-RSA, 2003.

Pedersen DKG: Overview

- Each participant *i* acts as the dealer and performs a Shamir secret sharing of a secret α_i, distributing shares w_{ij} to each other.
- The group's secret at the end is $sk \leftarrow \sum \alpha_i = \sum \sum w_{ij} \lambda_i$.
- Proven secure in the context of threshold signatures. [3]
- A rushing adversary *can* bias key material, but by a limited amount.

[3] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, Tal Rabin. Secure Applications of Pedersen's Distributed Key Generation Protocol, CT-RSA, 2003.

Pedersen DKG

Participant i **Other Participants**

$\alpha_i \leftarrow\!\$\ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\!\$\ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

Broadcast $\vec{D}_i$ $\longrightarrow$

Send w_{ij} $\longrightarrow$

$\text{Shamir.Verify}(w_i, \vec{D}_j) \stackrel{?}{=} 1$

$sk_i = \sum_{k \in \text{qual}} w_k \lambda_k,$ // qual *are the players that issued valid shares.*

$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$

Pedersen DKG

Participant i **Other Participants**

$\alpha_i \leftarrow\!\!\$\ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\!\!\$\ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

Broadcast $\vec{D}_i$ →

Send w_{ij} →

$\text{Shamir.Verify}(w_i, \vec{D}_j) \stackrel{?}{=} 1$

$sk_i = \sum_{k \in \text{qual}} w_k \lambda_k$, // qual *are the players that issued valid shares.*

$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$

Pedersen DKG

Participant i **Other Participants**

$\alpha_i \xleftarrow{\$} \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \xleftarrow{\$} \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

$\xrightarrow{\textbf{Broadcast } \vec{D}_i}$

$\xrightarrow{\textbf{Send } w_{ij}}$

$\text{Shamir.Verify}(w_i, \vec{D}_j) \stackrel{?}{=} 1$

$sk_i = \sum_{k \in \text{qual}} w_k \lambda_k,$ // qual *are the players that issued valid shares.*

$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$

Pedersen DKG

Participant i **Other Participants**

$\alpha_i \leftarrow\!\$\, \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\!\$\, \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

$\xrightarrow{\textbf{Broadcast } \vec{D}_i}$

$\xrightarrow{\textbf{Send } w_{ij}}$

$\text{Shamir.Verify}(w_i, \vec{D}_j) \stackrel{?}{=} 1$

$sk_i = \sum_{k \in \text{qual}} w_k \lambda_k,$ // qual *are the players that issued valid shares.*

$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$

Pedersen DKG

Participant i

Other Participants

$\alpha_i \leftarrow\$ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\$ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

Broadcast $\vec{D}_i$ $\longrightarrow$

Send w_{ij} $\longrightarrow$

$\text{Shamir.Verify}(w_i, \vec{D}_j) \stackrel{?}{=} 1$

$sk_i = \sum_{k \in \text{qual}} w_k \lambda_k$, // qual *are the players that issued valid shares.*

$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$

Pedersen DKG

Participant i | **Other Participants**

$\alpha_i \xleftarrow{\$} \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \xleftarrow{\$} \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

Broadcast $\vec{D}_i$ $\longrightarrow$

Send w_{ij} $\longrightarrow$

$\text{Shamir.Verify}(w_i, \vec{D}_j) \stackrel{?}{=} 1$

$sk_i = \sum_{k \in \text{qual}} w_k \lambda_k, \quad$ // qual *are the players that issued valid shares.*

$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$

Pedersen DKG

Participant i | **Other Participants**

$\alpha_i \leftarrow\$ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\$ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

$$\xrightarrow{\textbf{Broadcast } \vec{D}_i}$$

$$\xrightarrow{\textbf{Send } w_{ij}}$$

$\text{Shamir.Verify}(w_i, \vec{D}_j) \stackrel{?}{=} 1$

$sk_i = \sum_{k \in \text{qual}} w_k \lambda_k,$ // qual *are the players that issued valid shares.*

$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$

Pedersen DKG

Participant i | **Other Participants**

$\alpha_i \leftarrow\!\$\ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\!\$\ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

$\xrightarrow{\textbf{Broadcast } \vec{D}_i}$

$\xrightarrow{\textbf{Send } w_{ij}}$

$\text{Shamir.Verify}(w_i, \vec{D}_j) \stackrel{?}{=} 1$

$sk_i = \sum_{k \in \text{qual}} w_k \lambda_k$, // qual *are the players that issued valid shares.*

$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$

Key Bias Attack Against Pedersen DKG

- A rushing adversary can learn the output public key before publishing their own contribution.
- Nothing prevents this adversary from adaptively choosing their contributions.
- We show an example of this adversary forcing pk to be even without detection from other participants.

Key Bias Attack Against Pedersen DKG

- A rushing adversary can learn the output public key before publishing their own contribution.
- Nothing prevents this adversary from adaptively choosing their contributions.
- We show an example of this adversary forcing pk to be even without detection from other participants.

Key Bias Attack Against Pedersen DKG

- A rushing adversary can learn the output public key before publishing their own contribution.
- Nothing prevents this adversary from adaptively choosing their contributions.
- We show an example of this adversary forcing pk to be even without detection from other participants.

Key Bias Attack by a Rushing Adversary

Adversary **Other Participants**

Broadcast $\vec{D}_i$ ←

Send w_{ij} ←

Until pk is even, do:

$$\alpha_A \leftarrow_{\$} \mathbb{Z}_q; \quad \text{check } pk = g^{\alpha_A} \cdot \prod_{k=1; k\neq A}^{n} A_{k0}$$

$$(\{w_{A1}, \ldots, w_{An}\}, \{\alpha_A, a_{A1}, \ldots, a_{A(t-1)}\}) \leftarrow_{\$} \text{Shamir.Share}(\alpha_A, n, t)$$

$$\vec{D}_A = \langle A_{A0}, \ldots, A_{A(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_A, a_{A1}, \ldots, a_{A(t-1)})$$

Broadcast $\vec{D}_A$ →

Send w_{Aj} →

Key Bias Attack by a Rushing Adversary

Adversary **Other Participants**

Broadcast $\vec{D}_i$

Send w_{ij}

Until pk is even, do:

$$\alpha_A \leftarrow_{\$} \mathbb{Z}_q; \quad \text{check } pk = g^{\alpha_A} \cdot \prod_{k=1; k\neq A}^{n} A_{k0}$$

$$(\{w_{A1}, \ldots, w_{An}\}, \{\alpha_A, a_{A1}, \ldots, a_{A(t-1)}\}) \leftarrow_{\$} \text{Shamir.Share}(\alpha_A, n, t)$$

$$\vec{D}_A = \langle A_{A0}, \ldots, A_{A(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_A, a_{A1}, \ldots, a_{A(t-1)})$$

Broadcast $\vec{D}_A$

Send w_{Aj}

Key Bias Attack by a Rushing Adversary

Adversary | **Other Participants**

Broadcast $\vec{D}_i$ ←

Send w_{ij} ←

Until *pk* is even, do:

$$\alpha_A \leftarrow_{\$} \mathbb{Z}_q; \quad \text{check } pk = g^{\alpha_A} \cdot \prod_{k=1; k \neq A}^{n} A_{k0}$$

$$(\{w_{A1}, \ldots, w_{An}\}, \{\alpha_A, a_{A1}, \ldots, a_{A(t-1)}\}) \leftarrow_{\$} \text{Shamir.Share}(\alpha_A, n, t)$$

$$\vec{D}_A = \langle A_{A0}, \ldots, A_{A(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_A, a_{A1}, \ldots, a_{A(t-1)})$$

Broadcast $\vec{D}_A$ →

Send w_{Aj} →

Key Bias Attack by a Rushing Adversary

Adversary | **Other Participants**

Broadcast $\vec{D}_i$ ←

Send w_{ij} ←

Until *pk* is even, do:

$$\alpha_A \leftarrow_\$ \mathbb{Z}_q; \quad \text{check } pk = g^{\alpha_A} \cdot \prod_{k=1; k \neq A}^{n} A_{k0}$$

$$(\{w_{A1}, \ldots, w_{An}\}, \{\alpha_A, a_{A1}, \ldots, a_{A(t-1)}\}) \leftarrow_\$ \text{Shamir.Share}(\alpha_A, n, t)$$

$$\vec{D}_A = \langle A_{A0}, \ldots, A_{A(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_A, a_{A1}, \ldots, a_{A(t-1)})$$

Broadcast $\vec{D}_A$ →

Send w_{Aj} →

Key Bias Attack by a Rushing Adversary

Adversary | **Other Participants**

Broadcast $\vec{D}_i$ ←

Send w_{ij} ←

Until pk is even, do:

$$\alpha_A \leftarrow_{\$} \mathbb{Z}_q; \quad \text{check } pk = g^{\alpha_A} \cdot \prod_{k=1;k\neq A}^{n} A_{k0}$$

$$(\{w_{A1}, \ldots, w_{An}\}, \{\alpha_A, a_{A1}, \ldots, a_{A(t-1)}\}) \leftarrow_{\$} \text{Shamir.Share}(\alpha_A, n, t)$$

$$\vec{D}_A = \langle A_{A0}, \ldots, A_{A(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_A, a_{A1}, \ldots, a_{A(t-1)})$$

Broadcast $\vec{D}_A$ →

Send w_{Aj} →

Key Bias Attack by a Rushing Adversary

Adversary | **Other Participants**

Broadcast $\vec{D}_i$ ←

Send w_{ij} ←

Until pk is even, do:

$$\alpha_A \leftarrow_{\$} \mathbb{Z}_q; \quad \text{check } pk = g^{\alpha_A} \cdot \prod_{k=1; k \neq A}^{n} A_{k0}$$

$$(\{w_{A1}, \ldots, w_{An}\}, \{\alpha_A, a_{A1}, \ldots, a_{A(t-1)}\}) \leftarrow_{\$} \text{Shamir.Share}(\alpha_A, n, t)$$

$$\vec{D}_A = \langle A_{A0}, \ldots, A_{A(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_A, a_{A1}, \ldots, a_{A(t-1)})$$

Broadcast $\vec{D}_A$ →

Send w_{Aj} →

Key Bias Attack by a Rushing Adversary

Adversary **Other Participants**

Broadcast $\vec{D}_i$ $\longleftarrow$

Send w_{ij} $\longleftarrow$

Until *pk* is even, do:

$$\alpha_A \xleftarrow{\$} \mathbb{Z}_q; \quad \text{check } pk = g^{\alpha_A} \cdot \prod_{k=1; k \neq A}^{n} A_{k0}$$

$$(\{w_{A1}, \ldots, w_{An}\}, \{\alpha_A, a_{A1}, \ldots, a_{A(t-1)}\}) \xleftarrow{\$} \text{Shamir.Share}(\alpha_A, n, t)$$

$$\vec{D}_A = \langle A_{A0}, \ldots, A_{A(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_A, a_{A1}, \ldots, a_{A(t-1)})$$

Broadcast $\vec{D}_A$ $\longrightarrow$

Send w_{Aj} $\longrightarrow$

Key Bias Attack by a Rushing Adversary

Adversary **Other Participants**

Broadcast $\vec{D}_i$ ←

Send w_{ij} ←

Until *pk* is even, do:

$$\alpha_A \leftarrow\!\$\, \mathbb{Z}_q; \;\; \text{check } pk = g^{\alpha_A} \cdot \prod_{k=1;k\neq A}^{n} A_{k0}$$

$$(\{w_{A1}, \ldots, w_{An}\}, \{\alpha_A, a_{A1}, \ldots, a_{A(t-1)}\}) \leftarrow\!\$\, \text{Shamir.Share}(\alpha_A, n, t)$$

$$\vec{D}_A = \langle A_{A0}, \ldots, A_{A(t-1)} \rangle \leftarrow \text{Shamir.Commit}(\alpha_A, a_{A1}, \ldots, a_{A(t-1)})$$

Broadcast $\vec{D}_A$ →

Send w_{Aj} →

GJKR: Overview

- Stand-alone security; secure against key biasing.
- Assumes t honest players.
- Participants issue a blinded VSS commitment in round one and then unblinds in round three.
- Share correctness is verified using blinded commitments.
- Contributions from cheating players can be extracted in round three (by t honest players).

GJKR: Overview

- Stand-alone security; secure against key biasing.
- Assumes t honest players.
- Participants issue a blinded VSS commitment in round one and then unblinds in round three.
- Share correctness is verified using blinded commitments.
- Contributions from cheating players can be extracted in round three (by t honest players).

GJKR: Overview

- Stand-alone security; secure against key biasing.
- Assumes t honest players.
- Participants issue a blinded VSS commitment in round one and then unblinds in round three.
- Share correctness is verified using blinded commitments.
- Contributions from cheating players can be extracted in round three (by t honest players).

GJKR: Overview

- Stand-alone security; secure against key biasing.
- Assumes t honest players.
- Participants issue a blinded VSS commitment in round one and then unblinds in round three.
- Share correctness is verified using blinded commitments.
- Contributions from cheating players can be extracted in round three (by t honest players).

GJKR: Overview

- Stand-alone security; secure against key biasing.
- Assumes t honest players.
- Participants issue a blinded VSS commitment in round one and then unblinds in round three.
- Share correctness is verified using blinded commitments.
- Contributions from cheating players can be extracted in round three (by t honest players).

GJKR Construction: Rounds One and Two

Participant i **Other Participants**

$\alpha_i, b_i \leftarrow\$ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\$ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(g, \alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

// *Commit with base g*

$(\{z_{i1}, \ldots, z_{in}\}, \{b_i, \hat{a_{i1}}, \ldots, a_{i(\hat{t-1})}\}) \leftarrow\$ \text{Shamir.Share}(b_i, n, t)$

$\vec{E}_i = \langle \hat{A}_{i0}, \ldots, \hat{A}_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(h, b_i, \hat{a_{i1}}, \ldots, a_{i(\hat{t-1})})$

// *Commit with base h*

$\vec{H}_i \leftarrow \langle (A_{i0}\hat{A}_{i0}), \ldots, (A_{i(t-1)}\hat{A}_{i(t-1)}) \rangle$ // *Pedersen commitment*

Broadcast $\vec{H}_i$ $\longrightarrow$

Send w_{ij}, z_{ij} to player j $\longrightarrow$

GJKR Construction: Rounds One and Two

Participant i | **Other Participants**

$\alpha_i, b_i \leftarrow\$ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\$ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(g, \alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

// *Commit with base g*

$(\{z_{i1}, \ldots, z_{in}\}, \{b_i, \hat{a_{i1}}, \ldots, a_{i(\hat{t-1})}\}) \leftarrow\$ \text{Shamir.Share}(b_i, n, t)$

$\vec{E}_i = \langle \hat{A}_{i0}, \ldots, \hat{A}_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(h, b_i, \hat{a_{i1}}, \ldots, a_{i(\hat{t-1})})$

// *Commit with base h*

$\vec{H}_i \leftarrow \langle (A_{i0}\hat{A}_{i0}), \ldots, (A_{i(t-1)}\hat{A}_{i(t-1)}) \rangle$ // *Pedersen commitment*

$\xrightarrow{\textbf{Broadcast } \vec{H}_i}$

$\xrightarrow{\textbf{Send } w_{ij}, z_{ij} \text{ to player } j}$

GJKR Construction: Rounds One and Two

Participant i | **Other Participants**

$\alpha_i, b_i \leftarrow\$ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\$ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(g, \alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

// *Commit with base g*

$(\{z_{i1}, \ldots, z_{in}\}, \{b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}}\}) \leftarrow\$ \text{Shamir.Share}(b_i, n, t)$

$\vec{E}_i = \langle \hat{A}_{i0}, \ldots, \hat{A}_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(h, b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}})$

// *Commit with base h*

$\vec{H}_i \leftarrow \langle (A_{i0}\hat{A}_{i0}), \ldots, (A_{i(t-1)}\hat{A}_{i(t-1)}) \rangle$ // *Pedersen commitment*

$\xrightarrow{\textbf{Broadcast } \vec{H}_i}$

$\xrightarrow{\textbf{Send } w_{ij}, z_{ij} \text{ to player } j}$

GJKR Construction: Rounds One and Two

Participant i | **Other Participants**

$\alpha_i, b_i \leftarrow\$ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\$ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(g, \alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

// *Commit with base g*

$(\{z_{i1}, \ldots, z_{in}\}, \{b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}}\}) \leftarrow\$ \text{Shamir.Share}(b_i, n, t)$

$\vec{E}_i = \langle \hat{A}_{i0}, \ldots, \hat{A}_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(h, b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}})$

// *Commit with base h*

$\vec{H}_i \leftarrow \langle (A_{i0}\hat{A}_{i0}), \ldots, (A_{i(t-1)}\hat{A}_{i(t-1)}) \rangle$ // *Pedersen commitment*

Broadcast $\vec{H}_i$ $\longrightarrow$

Send w_{ij}, z_{ij} **to player** j $\longrightarrow$

GJKR Construction: Rounds One and Two

Participant i | **Other Participants**

$\alpha_i, b_i \leftarrow\!\$\ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\!\$\ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(g, \alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

// *Commit with base g*

$(\{z_{i1}, \ldots, z_{in}\}, \{b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}}\}) \leftarrow\!\$\ \text{Shamir.Share}(b_i, n, t)$

$\vec{E}_i = \langle \hat{A}_{i0}, \ldots, \hat{A}_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(h, b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}})$

// *Commit with base h*

$\vec{H}_i \leftarrow \langle (A_{i0}\hat{A}_{i0}), \ldots, (A_{i(t-1)}\hat{A}_{i(t-1)}) \rangle$ // *Pedersen commitment*

Broadcast $\vec{H}_i$ $\longrightarrow$

Send w_{ij}, z_{ij} to player j $\longrightarrow$

GJKR Construction: Rounds One and Two

Participant i | **Other Participants**

$\alpha_i, b_i \leftarrow\$ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\$ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(g, \alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

// *Commit with base g*

$(\{z_{i1}, \ldots, z_{in}\}, \{b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}}\}) \leftarrow\$ \text{Shamir.Share}(b_i, n, t)$

$\vec{E}_i = \langle \hat{A}_{i0}, \ldots, \hat{A}_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(h, b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}})$

// *Commit with base h*

$\vec{H}_i \leftarrow \langle (A_{i0}\hat{A}_{i0}), \ldots, (A_{i(t-1)}\hat{A}_{i(t-1)}) \rangle$ // *Pedersen commitment*

Broadcast $\vec{H}_i$ $\longrightarrow$

Send w_{ij}, z_{ij} to player j $\longrightarrow$

GJKR Construction: Rounds One and Two

Participant i | **Other Participants**

$\alpha_i, b_i \leftarrow\!\$\ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\!\$\ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(g, \alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

// *Commit with base g*

$(\{z_{i1}, \ldots, z_{in}\}, \{b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}}\}) \leftarrow\!\$\ \text{Shamir.Share}(b_i, n, t)$

$\vec{E}_i = \langle \hat{A}_{i0}, \ldots, \hat{A}_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(h, b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}})$

// *Commit with base h*

$\vec{H}_i \leftarrow \langle (A_{i0}\hat{A}_{i0}), \ldots, (A_{i(t-1)}\hat{A}_{i(t-1)}) \rangle$ // *Pedersen commitment*

Broadcast $\vec{H}_i$ $\longrightarrow$

Send w_{ij}, z_{ij} to player j $\longrightarrow$

GJKR Construction: Rounds One and Two

Participant i | **Other Participants**

$\alpha_i, b_i \xleftarrow{\$} \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \xleftarrow{\$} \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(g, \alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

// *Commit with base g*

$(\{z_{i1}, \ldots, z_{in}\}, \{b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}}\}) \xleftarrow{\$} \text{Shamir.Share}(b_i, n, t)$

$\vec{E}_i = \langle \hat{A}_{i0}, \ldots, \hat{A}_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(h, b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}})$

// *Commit with base h*

$\vec{H}_i \leftarrow \langle (A_{i0}\hat{A}_{i0}), \ldots, (A_{i(t-1)}\hat{A}_{i(t-1)}) \rangle$ // *Pedersen commitment*

$\xrightarrow{\textbf{Broadcast } \vec{H}_i}$

$\xrightarrow{\textbf{Send } w_{ij}, z_{ij} \text{ to player } j}$

GJKR Construction: Rounds One and Two

Participant i | **Other Participants**

$\alpha_i, b_i \leftarrow\$ \mathbb{Z}_q$

$(\{w_{i1}, \ldots, w_{in}\}, \{\alpha_i, a_{i1}, \ldots, a_{i(t-1)}\}) \leftarrow\$ \text{Shamir.Share}(\alpha_i, n, t)$

$\vec{D}_i = \langle A_{i0}, \ldots, A_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(g, \alpha_i, a_{i1}, \ldots, a_{i(t-1)})$

// *Commit with base g*

$(\{z_{i1}, \ldots, z_{in}\}, \{b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}}\}) \leftarrow\$ \text{Shamir.Share}(b_i, n, t)$

$\vec{E}_i = \langle \hat{A}_{i0}, \ldots, \hat{A}_{i(t-1)} \rangle \leftarrow \text{Shamir.Commit}(h, b_i, \hat{a_{i1}}, \ldots, \hat{a_{i(t-1)}})$

// *Commit with base h*

$\vec{H}_i \leftarrow \langle (A_{i0}\hat{A}_{i0}), \ldots, (A_{i(t-1)}\hat{A}_{i(t-1)}) \rangle$ // *Pedersen commitment*

$$\xrightarrow{\textbf{Broadcast } \vec{H}_i}$$

$$\xrightarrow{\textbf{Send } w_{ij}, z_{ij} \text{ to player } j}$$

GJKR Construction: Round Three

Participant i | **Other Participants**

Broadcast $\vec{D}_i$ $\longrightarrow$

excluded $\leftarrow \emptyset$

For all j where Shamir.Verify$(w_{ji}, \vec{D}_j) \neq 1$:

excluded $\leftarrow$ excluded $\cup \{j\}$

$$sk_i = \sum_{j \in \text{excluded}} \alpha + \sum_{k \in (\text{qual} \setminus \text{excluded})} w_k \lambda_k$$

$$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$$

GJKR Construction: Round Three

Participant i | **Other Participants**

Broadcast $\vec{D}_i$ $\longrightarrow$

excluded $\leftarrow \emptyset$

For all j where Shamir.Verify$(w_{ji}, \vec{D}_j) \neq 1$:

$$\text{excluded} \leftarrow \text{excluded} \cup \{j\}$$

$$sk_i = \sum_{j \in \text{excluded}} \alpha + \sum_{k \in (\text{qual} \setminus \text{excluded})} w_k \lambda_k$$

$$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$$

GJKR Construction: Round Three

Participant i | **Other Participants**

$\xrightarrow{\textbf{Broadcast } \vec{D}_i}$

excluded $\leftarrow \emptyset$

For all j where Shamir.Verify$(w_{ji}, \vec{D}_j) \neq 1$:

excluded $\leftarrow$ excluded $\cup \{j\}$

$$sk_i = \sum_{j \in \text{excluded}} \alpha + \sum_{k \in (\text{qual} \setminus \text{excluded})} w_k \lambda_k$$

$$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$$

GJKR Construction: Round Three

Participant i | **Other Participants**

Broadcast $\vec{D}_i$ $\longrightarrow$

excluded $\leftarrow \emptyset$

For all j where Shamir.Verify$(w_{ji}, \vec{D}_j) \neq 1$:

excluded $\leftarrow$ excluded $\cup \{j\}$

$$sk_i = \sum_{j \in \text{excluded}} \alpha + \sum_{k \in (\text{qual} \setminus \text{excluded})} w_k \lambda_k$$

$$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$$

GJKR Construction: Round Three

Participant i | **Other Participants**

Broadcast $\vec{D}_i$ $\longrightarrow$

excluded $\leftarrow \emptyset$

For all j where Shamir.Verify$(w_{ji}, \vec{D}_j) \neq 1$:

excluded $\leftarrow$ excluded $\cup \{j\}$

$$sk_i = \sum_{j \in \text{excluded}} \alpha + \sum_{k \in (\text{qual} \setminus \text{excluded})} w_k \lambda_k$$

$$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$$

GJKR Construction: Round Three

Participant i | **Other Participants**

$$\xrightarrow{\textbf{Broadcast } \vec{D}_i}$$

excluded $\leftarrow \emptyset$

For all j where Shamir.Verify$(w_{ji}, \vec{D}_j) \neq 1$:

$$\text{excluded} \leftarrow \text{excluded} \cup \{j\}$$

$$sk_i = \sum_{j \in \text{excluded}} \alpha + \sum_{k \in (\text{qual} \setminus \text{excluded})} w_k \lambda_k$$

$$pk \leftarrow \prod_{k \in \text{qual}} A_{k0} = g^{\sum_{i \in \text{qual}} \alpha_i}$$

Uniform Distribution of Key Material in GJKR

- Cheating parties must choose their contributions having seen only the blinded commitment $\vec{H}_i$ from other players.
- Pedersen commitments guarantee that $H_i = g^i h^j$ will not reveal any information about g^i.
- If player A cheats and is kicked out in round 3, α_A can be extracted (assuming t honest parties).
- So key material will remain unbiased even if a player cheats in the last round after learning pk.

Uniform Distribution of Key Material in GJKR

- Cheating parties must choose their contributions having seen only the blinded commitment $\vec{H}_i$ from other players.
- Pedersen commitments guarantee that $H_i = g^i h^j$ will not reveal any information about g^i.
- If player A cheats and is kicked out in round 3, α_A can be extracted (assuming t honest parties).
- So key material will remain unbiased even if a player cheats in the last round after learning pk.

Uniform Distribution of Key Material in GJKR

- Cheating parties must choose their contributions having seen only the blinded commitment $\vec{H}_i$ from other players.
- Pedersen commitments guarantee that $H_i = g^i h^j$ will not reveal any information about g^i.
- If player A cheats and is kicked out in round 3, α_A can be extracted (assuming t honest parties).
- So key material will remain unbiased even if a player cheats in the last round after learning pk.

Uniform Distribution of Key Material in GJKR

- Cheating parties must choose their contributions having seen only the blinded commitment $\vec{H}_i$ from other players.

- Pedersen commitments guarantee that $H_i = g^i h^j$ will not reveal any information about g^i.

- If player A cheats and is kicked out in round 3, α_A can be extracted (assuming t honest parties).

- So key material will remain unbiased even if a player cheats in the last round after learning pk.

Takeaways

- DKGs are a useful building block for distributing trust among set of parties.
- Properties for centralized protocols are difficult to guarantee in a multi-party setting (such as uniform distribution of key material).
- Pedersen DKG is simple and efficient, but must be proven seperately for each context which it is used.
- GJKR is proven in a stand-alone manner, but at the loss of efficiency.
- Stay tuned for future research!

Takeaways

- DKGs are a useful building block for distributing trust among set of parties.
- Properties for centralized protocols are difficult to guarantee in a multi-party setting (such as uniform distribution of key material).
- Pedersen DKG is simple and efficient, but must be proven seperately for each context which it is used.
- GJKR is proven in a stand-alone manner, but at the loss of efficiency.
- Stay tuned for future research!

Takeaways

- DKGs are a useful building block for distributing trust among set of parties.
- Properties for centralized protocols are difficult to guarantee in a multi-party setting (such as uniform distribution of key material).
- Pedersen DKG is simple and efficient, but must be proven seperately for each context which it is used.
- GJKR is proven in a stand-alone manner, but at the loss of efficiency.
- Stay tuned for future research!

Takeaways

- DKGs are a useful building block for distributing trust among set of parties.
- Properties for centralized protocols are difficult to guarantee in a multi-party setting (such as uniform distribution of key material).
- Pedersen DKG is simple and efficient, but must be proven seperately for each context which it is used.
- GJKR is proven in a stand-alone manner, but at the loss of efficiency.
- Stay tuned for future research!

Takeaways

- DKGs are a useful building block for distributing trust among set of parties.
- Properties for centralized protocols are difficult to guarantee in a multi-party setting (such as uniform distribution of key material).
- Pedersen DKG is simple and efficient, but must be proven seperately for each context which it is used.
- GJKR is proven in a stand-alone manner, but at the loss of efficiency.
- Stay tuned for future research!

Thank you!