Fast batched asynchronous distributed key generation

Jens Groth (Nexus Labs)
Victor Shoup (Offchain Labs)

Motivation: threshold Schnorr Signatures

- Asynchronous communication model
- Optimal resilience
  - Parties $P_1, \ldots, P_n$, at most $t < n/3$ may be corrupt
- Robust (guaranteed output delivery)
- Offline/online paradigm
  - high throughput in both offline and online phases
  - low latency in online phase

New protocol

- **Linear** amortized (optimistic) communication complexity (in both phases)
- Amortized computational complexity: $O(n + \lambda/n)$ group *additions*
  - λ = bit length of group order

Motivation: threshold Schnorr Signatures

- Asynchronous communication model
- Optimal resilience
  - Parties $P_1, \ldots, P_n$, at most $t < n/3$ may be corrupt
- Robust (guaranteed output delivery)
- Offline/online paradigm
  - high throughput in both offline and online phases
  - low latency in online phase

New protocol

- **Linear** amortized (optimistic) communication complexity (in both phases)
- Amortized computational complexity: $O(n + \lambda/n)$ group *additions*
  - λ = bit length of group order

Motivation: threshold Schnorr Signatures

- Asynchronous communication model
- Optimal resilience
  - Parties $P_1, \ldots, P_n$, at most $t < n/3$ may be corrupt
- Robust (guaranteed output delivery)
- Offline/online paradigm
  - high throughput in both offline and online phases
  - low latency in online phase

New protocol

- **Linear** amortized (optimistic) communication complexity (in both phases)
- Amortized computational complexity: $O(n + \lambda/n)$ group *additions*
  - λ = bit length of group order

Motivation: threshold Schnorr Signatures

- Asynchronous communication model
- Optimal resilience
  - Parties $P_1, \ldots, P_n$, at most $t < n/3$ may be corrupt
- Robust (guaranteed output delivery)
- Offline/online paradigm
  - high throughput in both offline and online phases
  - low latency in online phase

New protocol

- **Linear** amortized (optimistic) communication complexity (in both phases)
- Amortized computational complexity: $O(n + \lambda/n)$ group *additions*
  - λ = bit length of group order

Motivation: threshold Schnorr Signatures

- Asynchronous communication model
- Optimal resilience
  - Parties $P_1, \ldots, P_n$, at most $t < n/3$ may be corrupt
- Robust (guaranteed output delivery)
- Offline/online paradigm
  - high throughput in both offline and online phases
  - low latency in online phase

New protocol

- **Linear** amortized (optimistic) communication complexity (in both phases)
- Amortized computational complexity: $O(n + \lambda/n)$ group *additions*
  - λ = bit length of group order

Motivation: threshold Schnorr Signatures

- Asynchronous communication model
- Optimal resilience
  - Parties $P_1, \ldots, P_n$, at most $t < n/3$ may be corrupt
- Robust (guaranteed output delivery)
- Offline/online paradigm
  - high throughput in both offline and online phases
  - low latency in online phase

New protocol

- **Linear** amortized (optimistic) communication complexity (in both phases)
- Amortized computational complexity: $O(n + \lambda/n)$ group *additions*
  - λ = bit length of group order

Motivation: threshold Schnorr Signatures

- Asynchronous communication model
- Optimal resilience
  - Parties $P_1, \ldots, P_n$, at most $t < n/3$ may be corrupt
- Robust (guaranteed output delivery)
- Offline/online paradigm
  - high throughput in both offline and online phases
  - low latency in online phase

New protocol

- **Linear** amortized (optimistic) communication complexity (in both phases)
- Amortized computational complexity: $O(n + \lambda/n)$ group *additions*
  - λ = bit length of group order

Example

$$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

\# of group additions/sig (both phases): 23 $\Longrightarrow$ $5\mu s$

other overheads (erasure coding / hashing / scalar-ops): $5\mu s$

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput *. . . even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ $5\mu s$

other overheads (erasure coding / hashing / scalar-ops): $5\mu s$

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput *. . . even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}$, $\lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ $5\mu s$

other overheads (erasure coding / hashing / scalar-ops): $5\mu s$

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput *. . . even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ 5μs

other overheads (erasure coding / hashing / scalar-ops): 5μs

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput . . . *even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}$, $\lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

\# of group additions/sig (both phases): 23 $\implies$ 5μs

other overheads (erasure coding / hashing / scalar-ops): 5μs

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\implies$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\implies$ better CPU utilization + faster algorithms $\implies$ better *overall* throughput *. . . even when demand never lets up*

Example

$$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ $5\mu s$

other overheads (erasure coding / hashing / scalar-ops): $5\mu s$

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput *...even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

\# of group additions/sig (both phases): 23 $\Longrightarrow$ $5\mu s$

other overheads (erasure coding / hashing / scalar-ops): $5\mu s$

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput . . . *even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ 5μs

other overheads (erasure coding / hashing / scalar-ops): 5μs

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput ... *even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ 5μs

other overheads (erasure coding / hashing / scalar-ops): 5μs

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput . . . *even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ $5\mu s$

other overheads (erasure coding / hashing / scalar-ops): $5\mu s$

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput . . . *even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ 5μs

other overheads (erasure coding / hashing / scalar-ops): 5μs

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$

better *overall* throughput . . . *even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ 5μs

other overheads (erasure coding / hashing / scalar-ops): 5μs

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput . . . *even when demand never lets up*

Example

$n = 49 = \underbrace{3 \cdot 16 + 1}_{3t+1}, \quad \lambda = 256$ [secp256k1]

micro-benchmarks: Macbook Pro / Apple M1 Max / single thread

# of group additions/sig (both phases): 23 $\Longrightarrow$ 5μs

other overheads (erasure coding / hashing / scalar-ops): 5μs

comms per party (both phases): 24 scalars, 9 grp elts

independent of n

$\Longrightarrow$ throughput **100K sig/sec** (both phases)

assumes 1Gb/s network bandwidth

Key advantages of offline/online paradigm

Lower latency in the online phase

Batch computation $\Longrightarrow$ better CPU utilization + faster algorithms $\Longrightarrow$ better *overall* throughput *. . . even when demand never lets up*

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

- (with completeness)
- Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

- Dealer shares a polynomial f — each P_i gets $f(i)$ *and* $f(0)\mathcal{G} \in E$
- Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

- Reduction from batch GoAVSS to batch AVSS
  - No poly commitments
  - Based on a *simple statistical test* — more efficient as n gets larger (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)
- Plug in batch AVSS of [Shoup & Smart 2023]:
  - Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

- (with completeness)
- Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

- Dealer shares a polynomial f — each P_i gets $f(i)$ *and* $f(0)\mathcal{G} \in E$
- Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

- Reduction from batch GoAVSS to batch AVSS
  - No poly commitments
  - Based on a *simple statistical test* — more efficient as n gets larger (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)
- Plug in batch AVSS of [Shoup & Smart 2023]:
  - Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

- (with completeness)
- Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

- Dealer shares a polynomial f — each P_i gets $f(i)$ *and $f(0)\mathcal{G} \in E$*
- Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

- Reduction from batch GoAVSS to batch AVSS
  - No poly commitments
  - Based on a *simple statistical test* — more efficient as n gets larger
    (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)
- Plug in batch AVSS of [Shoup & Smart 2023]:
  - Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

- (with completeness)
- Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

- Dealer shares a polynomial f — each P_i gets $f(i)$ $\underline{\textit{and } f(0)\mathcal{G} \in E}$
- Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

- Reduction from batch GoAVSS to batch AVSS
  - No poly commitments
  - Based on a *simple statistical test* — more efficient as n gets larger
    (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)
- Plug in batch AVSS of [Shoup & Smart 2023]:
  - Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

(with completeness)

Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

Dealer shares a polynomial f — each P_i gets $f(i)$ *and* $f(0)\mathcal{G} \in E$

Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

Reduction from batch GoAVSS to batch AVSS

No poly commitments

Based on a *simple statistical test* — more efficient as n gets larger (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)

Plug in batch AVSS of [Shoup & Smart 2023]:

Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

(with completeness)

Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

Dealer shares a polynomial f — each P_i gets $f(i)$ *and* $f(0)\mathcal{G} \in E$

Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

Reduction from batch GoAVSS to batch AVSS

No poly commitments

Based on a *simple statistical test* — more efficient as n gets larger (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)

Plug in batch AVSS of [Shoup & Smart 2023]:

Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

(with completeness)

Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

Dealer shares a polynomial f — each P_i gets $f(i)$ and $f(0)\mathcal{G} \in E$

Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

Reduction from batch GoAVSS to batch AVSS

No poly commitments

Based on a *simple statistical test* — more efficient as n gets larger
(up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)

Plug in batch AVSS of [Shoup & Smart 2023]:

Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

(with completeness)

Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

Dealer shares a polynomial f — each P_i gets $f(i)$ *and $f(0)\mathcal{G} \in E$*

Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

- Reduction from batch GoAVSS to batch AVSS
  - No poly commitments
  - Based on a *simple statistical test* — more efficient as n gets larger (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)
- Plug in batch AVSS of [Shoup & Smart 2023]:
  - Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

(with completeness)

Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

Dealer shares a polynomial f — each P_i gets $f(i)$ *and* $f(0)\mathcal{G} \in E$

Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

Reduction from batch GoAVSS to batch AVSS

No poly commitments

Based on a *simple statistical test* — more efficient as n gets larger (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)

Plug in batch AVSS of [Shoup & Smart 2023]:

Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

- (with completeness)
- Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

- Dealer shares a polynomial f — each P_i gets $f(i)$ *and* $f(0)\mathcal{G} \in E$
- Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

- Reduction from batch GoAVSS to batch AVSS
  - No poly commitments
  - Based on a *simple statistical test* — more efficient as n gets larger (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)
- Plug in batch AVSS of [Shoup & Smart 2023]:
  - Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 1: a better batch GoAVSS protocol

AVSS: asynchronous verifiable secret sharing

(with completeness)

Dealer shares a polynomial f — each P_i (eventually) gets $f(i)$

GoAVSS: Group-oriented AVSS (over a group $E = \langle \mathcal{G} \rangle$)

Dealer shares a polynomial f — each P_i gets $f(i)$ *and* $f(0)\mathcal{G} \in E$

Allows a dealer to "publish a PK" and "share a SK"

Batch (Go)AVSS: dealer shares many polys (at the same time)

A new Batch GoAVSS Protocol:

- Reduction from batch GoAVSS to batch AVSS
  - No poly commitments
  - Based on a *simple statistical test* — more efficient as n gets larger (up to a point: $O(\lambda/n)$ grp ops $+O(n)$ scalar ops)
- Plug in batch AVSS of [Shoup & Smart 2023]:
  - Linear amortized (optimistic) comms complexity / lightweight crypto

Main Result 2: a better batch randomness extractor

- Basic idea: generate a batch of "published PKs" and "shared SKs"
  - Each party runs GoAVSS as dealer
  - Agree on a set of $\frac{2}{3}n$ dealings
  - Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**
- A brief history of super invertible matrices
  - Ancient history: coding theory — generator matrix for MDS code
  - MPC — [Hirt & Nielsen 2006]
  - Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]
- **Additive complexity** of super-invertible matrix / vector multiplication
  - # of grp ops for randomness extraction (amortized per grp elt output)
  - Naive: $O(\lambda n)$
  - FFT: $O(\lambda \log n)$
  - Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
  - **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
    - [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of “published PKs” and “shared SKs”

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner’s rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a
  super invertible matrix

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of “published PKs” and “shared SKs”

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner’s rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  - [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of “published PKs” and “shared SKs”

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner’s rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  - [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]

Main Result 2: a better batch randomness extractor

Basic idea: generate a batch of "published PKs" and "shared SKs"

- Each party runs GoAVSS as dealer
- Agree on a set of $\frac{2}{3}n$ dealings
- Extract $\frac{1}{3}n$ random(ish) dealings by linearly combining dealings with a **super invertible matrix**

A brief history of super invertible matrices

- Ancient history: coding theory — generator matrix for MDS code
- MPC — [Hirt & Nielsen 2006]
- Threshold PK crypto — SPRINT [Benhamouda, Halevi, Krawczyk, Ma, Rabin 2023]

Additive complexity of super-invertible matrix / vector multiplication

- # of grp ops for randomness extraction (amortized per grp elt output)
- Naive: $O(\lambda n)$
- FFT: $O(\lambda \log n)$
- Horner's rule: $O(n \log n)$ [better than FFT when $n \ll \lambda$]
- **Our observation — Pascal matrix:** $n/2$ (and sometimes $n/3$)
  [for $\lambda = 256$, should be much better than FFT for n up to at least $n \approx 1000$]