Fast SNARK-based Non-Interactive Distributed Verifiable Random Function with Ethereum Compatibility

Jia Liu

INPUT | OUTPUT

* work done while at Enya Labs

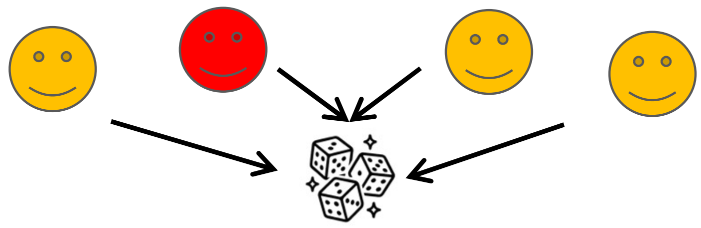
Mark Manulis

PACY Lab @ RI CODE
Universität der Bundeswehr München

ACM AsiaCCS 2025
25-29 August in Hanoi, Vietnam

Distributed Randomness Generation

- aka **Distributed Randomness Beacons** (see survey IEEE S&P'23)
- Pseudorandom generator based on contributions of n *different* sources
- Not all sources need to be trusted: *t-out-of-n* trust model
- Third parties should be able to *verify* the outputs prior to using them



Existing approaches for DRBs

- **Leader-based election protocols**
  - New leader per round uses VRF to output a random beacon
  - Example: Algorand, Ouroboros-Praos, Elrond
  - Withholding attack, i.e. leader may refuse to provide random output

- **Commit-Reveal(-Recover) protocols**
  - Every party commits to randomness, which is then revealed and aggregated
  - Last-revealer attack, i.e. last party to reveal may refuse to do so
  - Can be mitigated using DKG or PVSS – introduces extra complexity overheads

- **Verifiable Delay Function based protocols**
  - VDFs ensure that the output is released after a predetermined period of time
  - (Non-cryptographic) trust assumption, typically assuming expensive hardware (ASICs)
  - Often subject to parallel computation attacks, e.g. against Minroot

Approaches based on Distributed VRFs

- **DVRF: Distributed VRF** = t-out-of-n VRF
  - Stage 1: Parties run a DKG protocol to compute pk and own shares sk_i
  - Stage 2: Parties use sk_i to generate aggregatable verifiable random shares

- **DVRFs with *interactive* DKGs**
  - Ex: Drand, HERB, DDH-DRB, Glow-DRB
  - Interactive DKGs generally introduce high overheads to be practical

- **DVRFs with *non-interactive* DKGs**
  - Groth21 NI-DKG uses costly *chunk encryption* and BLS12-381 curve
  - "DKG inside a SNARK" code from 2022 using BLS12-377 / BW6 curve

NI-DVRF syntax overview

$(msk_i, mpk_i) \longleftarrow \mathrm{KGen}(\kappa)$

initialisation

(msk_i, mpk_i)

$(QUAL, gpk, vk, sk) \longleftarrow \mathrm{NIDKG}(t, n, M)$

M : initial member set

QUAL : qualified member set

gpk : global public key

randomness generation

$(sk[i], vk[i])$

n

$\sigma_{x,i} \longleftarrow \mathrm{PartialEval}(x, sk[i])$

x : public DVRF input

$\sigma_{x,i}$

t valid $\sigma_{x,i}$

$0/1 \longleftarrow \mathrm{PartialVerify}(x, vk[i], \sigma_{x,i})$

$(v, \pi) \longleftarrow \mathrm{Combine}(vk, x, \{\sigma_{x,i}\}_i)$

$0/1 \longleftarrow \mathrm{Verify}(gpk, x, v, \pi)$

v : random DVRF output

π : public proof

NI-DVRF properties and security goals

Robustness: guaranteed output v in presence of up to t corrupted members

- NI-DVRF avoids costly resolution and requires only 1 message per party

Uniqueness: public input x deterministically determines the output v

- Crucial for many apps, e.g. next block proposer, validator sets, etc

Strong Pseudorandomness: distribution of v is random, implies **unpredictability**

- Strong = Adversary can query PartialEval oracle on challenge x up to t-1 times

Public verifiability: anyone can verify that v was computed correctly

- Eliminates the need to trust any party with honest generation of v

Our NI-DVRF highlights

PACY LAB

- Improves upon interactive Glow-DRB (Galindo et al, EuroS&P'21)

- Ingredients using type-3 pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$
  - SNARK-based NIDKG protocol
  - Threshold BLS signature for DVRF outputs

- Implementation compatible with Ethereum(-like) chains
  - Main protocol in Rust. Solidity contracts for Ethereum on-chain verification.
  - Adopts BN256 curve supported by Ethereum.

- PoC evaluation on Boba Network's DRB service zkRand.

boba

Our NI-DVRF scheme: Initialisation

- param : $(\mathbb{G}=\langle g\rangle, p), (e, \mathbb{G}_1=\langle g_1\rangle, \mathbb{G}_2=\langle g_2\rangle, q)$
- $H_1: \mathbb{G} \rightarrow \mathbb{Z}_q, H_2 : \{0,1\}^* \rightarrow \mathbb{G}_1, H_3 : \{0,1\}^* \rightarrow \mathbb{Z}_q, H_4: \mathbb{G}_1 \rightarrow \{0,1\}^*$

$(msk_i, g^{msk_i}) \longleftarrow KGen(\kappa)$

$(QUAL, gpk, vk, sk) \longleftarrow NIDKG(t, n, M)$

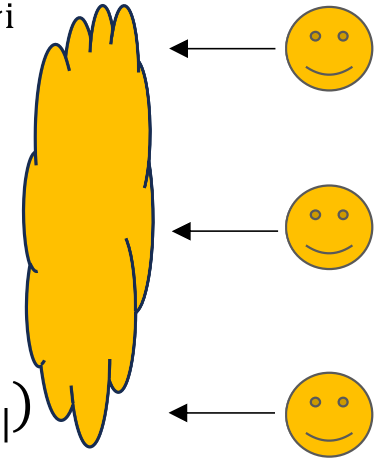
pp_i, enc_i , SNARK π_i

$f_i(x) = a_{i,0} + \ldots + a_{i,t-1}x^{t-1}$

$s_{i,1}= f_i(1), \ldots , s_{i,|M|}= f_i(|M|)$

$pp_i = (g_1^{s_{i,1}},\ldots, g_1^{s_{i,|M|}}, g_1^{a_{i,0}}, g_2^{a_{i,0}})$

$enc_i = (h = g^r, \{c_{i,j} = H_1(mpk_j^r) + s_{i,j}\}_{j\in|M|})$



$QUAL = \{i\}$ with valid π_i

$\{s_{j,i} = c_{j,i} - H_1(h^{msk_i})\}_{j\in QUAL}$

$sk[i] = \sum_{j\in QUAL} s_{j,i}$

$vk[i] = \prod_{j\in QUAL} g_1^{s_{j,i}} = g_1^{sk[i]}$

$gpk = \prod_{j\in QUAL} g_2^{a_{j,0}}$

(QUAL, gpk, vk) are publicly computable from all (pp_i, enc_i , SNARK π_i)

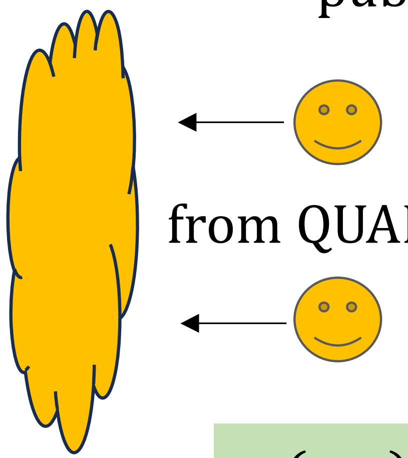
Our NI-DVRF scheme: Gen randomness

$\sigma_{x,i} \longleftarrow \text{PartialEval}(x, sk[i], vk[i])$

$\sigma_{x,i} = (i, v_i, \text{NIZK } \pi_i)$

$v_i = H_2(x)^{sk[i]}$

$\pi_i = \text{NIZK}[sk[i]] : DL(v_i) = DL(vk[i])$



$0/1 \longleftarrow \text{PartialVerify}(x, vk[i], \sigma_{x,i})$

public check of NIZK π_i

$(v, \pi) \longleftarrow \text{Combine}(vk, x, \{\sigma_{x,i}\}_i)$

public set $I \subseteq$ QUAL of t valid $\sigma_{x,i}$:

$\pi = \prod_{j\in I} v_j^{\lambda_j(0)}$

$v = H_4(\pi)$

$(v, \pi) \longleftarrow \text{Verify}(gpk, x, v, \pi)$

two public checks :

$e(\pi, g_2) \stackrel{?}{=} e(H_2(x), gpk)$

$v \stackrel{?}{=} H_4(\pi)$

Security of our NI-DVRF

- **Pseudorandomness** under co-CDH and SDH in ROM.
  - co-CDH : given $(g_1^{\alpha}, g_1^{\beta}, g_2^{\alpha})$ hard to compute $g_1^{\alpha\beta}$
  - SDH : given $(g, g^{\alpha}, g^{\beta})$ and oracle $O_{\beta}(U,X)$: $U^{\beta} \stackrel{?}{=} X$ hard to compute $g^{\alpha\beta}$

- **Strong pseudorandomness** under co-CDH and extended XDH assumption in ROM.
  - extended XDH : extended DDH (Agrawal et al, CCS'18) in $\mathbb{G}_1$

$$(g_1, g_1^{\alpha_1}, \dots, g_1^{\alpha_n}, g_1^{\beta}, g_1^{\alpha_1\beta}, \dots, g_1^{\alpha_n\beta})$$
$$\approx_c$$
$$(g_1, g_1^{\alpha_1}, \dots, g_1^{\alpha_n}, g_1^{\beta}, y_1, \dots, y_n) \text{ for } y_i \in_R \mathbb{G}_1$$

Implementation and optimisations I

- SNARK π : Halo2 with KZG commitment on BN256 curve

- DKG circuit proves (pp_i, enc_i) is computed correctly:
  - $pp_i = (g_1^{s_{i,1}}, \ldots, g_1^{s_{i,|M|}}, g_1^{a_{i,0}}, g_2^{a_{i,0}})$, $enc_i = (h = g^r, \{c_{i,j} = H_1(mpk_j^r) + s_{i,j}\}_{j \in |M|})$

- Public shares in pp_i from secret shares:
  - non-native encodings on BN256
  - optimised scalar-point mult gates leading to 70% reduction in gates

- Encryption of secret shares in enc_i:
  - on Grumpkin curve which has same base field $\mathbb{F}_q$ as BN256
  - native encodings on Grumpkin 25x smaller than non-native on BN256

Implementation and optimisations II

Smart contracts in Solidity for onchain verification and computation:

- Verification of SNARKs π_i for (pp_i, enc_i) in NI-DKG
- Computation of global public key gpk
- Verification of NIZKs π_i for partial evaluations v_i : $DL(v_i) \stackrel{?}{=} DL(vk[i])$
- Computation of final pseudorandom output (v, π)
- Verification of (v, π) : $e(\pi, g_2) \stackrel{?}{=} e(H_2(x), gpk)$ and $v \stackrel{?}{=} H_4(\pi)$

Code & Demo available at https://github.com/bobanetwork/zkrand

zkRand is a chosen name by Boba Network for our NI-DVRF

zkRand-NIDKG performance

- NIDKG on AWS instance r6i.8xlarge (32 CPUs, 256GB of RAM)

Circuit degree	t, n	Curve	Prove (s)	Verify (ms)	Proof size (B)	Dealing size (B)	Peak memory (GB)
18	(3, 5)	BN256	20.8	5.1	3488	448	4.8
20	(20, 38)		74.7	6.0		2560	16.5
22	(86, 171)		294.3	10.1		11072	64.4
					SNARK π	(pp_i , enc_i)	

- **Scalability**: typical blockchain applications 10 to 30 nodes
  - For large sets, divide into smaller subsets and rotate using random outputs
  - for example, 10 subsets each with 16 nodes instead of 160 nodes

zkRand-Randomness generation performance

Timings for

- Creating/verifying partial evaluations $\sigma_{x,i} = (i, v_i, \text{NIZK } \pi_i)$
- Combining t valid evaluations and verifying final output (v, π)

t, n	PartialEval (ms)	PartialVerify (ms)	Combine (ms)	Verify (ms)
(3, 5)	0.86	1.02	0.7	1.62
(20, 38)			4.2	
(86, 171)			18.5	

zkRand Gas cost for onchain deployment

Costs for on-chain verification on Etherum in Gas currency:

t, n	Verify SNARK π	PartialVerify	PartialVerify (fast*)	Verify	Verify (fast*)
(3, 5)	726115	101392	55098	193693	147468
(20, 38)	972917				
(86, 171)	1985415				

*fast : value $H_2(x)$ is computed once and stored in the contract

Lazy verification to save costs: deposit locked away for a specific period and is paid to anyone who challenges verification and finds that is invalid.

Comparing zkRand-NIDKG with selected DKGs

Scheme	t, n	Curve	Prove (s)	Verify (ms)	Proof size (B)	Dealing size (B)	Ethereum-compatible
zkRand-NIDKG	(3, 5)	**BN256**	20.8	**5.1**	3488	**448**	**Yes**
cdDKG (EC'24)	(3, 5)	BLS12-381*	0.2	153.4	383	1311	No
cgDKG (CCS'24)	(3, 5)	BLS12-381*	0.1	106.8	675	1460	No
Groth21	(3, 5)	BLS12-381	0.2	103.0	3770	7800	No
zkRand-NIDKG	(86, 171)	**BN256**	294.3	**10.1**	3488	**11072**	**Yes**
cdDKG (EC'24)	(86, 171)	BLS12-381*	1.5	1319.3	383	37634	No
cgDKG (CCS'24)	(86, 171)	BLS12-381*	0.5	650.5	675	41844	No
Groth21	(86, 171)	BLS12-381	4.9	2623.5	11904	220504	No

Summary

- NI-DVRF using SNARK-based NIDKG and Threshold BLS for non-interactive randomness generation

- (Strong) pseudorandomness / unpredictability, uniqueness, robustness, public verifiability

- Optimised implementation for Ethereum and Ethereum-like networks using the BN256 curve

Privacy and Applied Cryptography Lab

PACY

Analysis Design • Engineering

Mark Manulis

PACY Lab, Research Institute CODE

Universität der Bundeswehr München

mark.manulis@unibw.de

https://www.unibw.de/pacy-en