

Menshen and Octopus-(T)CL: Two-Party and Threshold ECDSA

NIST MPTC Preview Talk · Category N1 (ECDSA Sign)

Team Octopus: Man Ho Au[1], Robert H. Deng[3], Liming Gao[2], Bowen Jiang[3], Xianhui Lu[2], Tian Qiu[4], Guofeng Tang[3], Ruida Wang[2], **Haiyang Xue**[3], Guomin Yang[3], Chengru Zhang[5]

7 Oct 2026

[1]The Hong Kong Polytechnic University [2]IIE, CAS [3]Singapore Management University
[4]Nanyang Technological University [5]Ethereum Foundation

Presented at NIST TCPT3

Menshen and Octopus-(T)CL at a Glance

Menshen
two-party

2-out-of-n
1 corruption

one MtA/OLE
no key share inside

P2P only
no broadcast channel

online: 32 B
+ one signature verification

Octopus-CL
threshold, from CL

t-out-of-n
up to $t-1$ corruptions

GOD
identifiable abort + robustness

P2P + broadcast
to agree on who cheated

EC-only online
a DDH proof, no CL operations

Octopus-TCL
threshold, from threshold CL

t-out-of-n
up to $t-1$ corruptions

$O(1)$ message size
independent of n

P2P only
no broadcast channel

security with abort
threshold CL encryption

all three: **3 rounds** (2 offline + 1 online) · standard ECDSA output (FIPS 186-5)

Why Threshold ECDSA Is Hard

ECDSA Is Not Linear in Its Secrets

$$k \xleftarrow{\$} \mathbb{Z}_q, \quad R = kG, \quad r = R|_x \bmod q$$

$$s = k^{-1}(h + r\,x), \qquad h = H(m)$$

with k and x secret-shared, s needs shares of

$$k^{-1} \quad \text{and} \quad k^{-1}x$$

$\Rightarrow$ **products of secrets**, not sums



$$\alpha + \beta = a\,b \bmod q$$

Multiplicative-to-Additive conversion,
a.k.a. **Oblivious Linear Evaluation (OLE)**

instantiations:
OT · Paillier · CL · Joye–Libert

the **dominant cost**: $\approx$90 KB per OT call,
range proofs (Paillier, Joye-Libert),
class-group exponentiations (CL)

offline, message-independent: MtA/OLE, nonce R

online, after m: ideally one short message

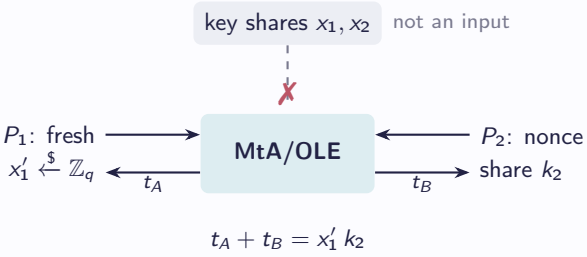
Three Settings, Three Targets

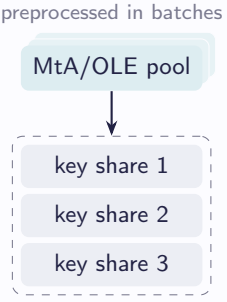
setting	bottleneck in prior work	our target
two-party 2-out-of-n	2–4 MtA/OLE calls, fed with key shares	**Menshen** one MtA/OLE, no key share inside
t-**out-of-**n **with GOD** identifiable abort + robustness	GOD costs 4–7 rounds	**Octopus-CL** GOD in 3 rounds, EC-only online
t-**out-of-**n from threshold CL	constant-size messages, but 4 rounds	**Octopus-TCL** constant-size messages, 3 rounds

Menshen: Two-Party ECDSA from One Key-Independent MtA/OLE

Motivation: One MtA/OLE, with No Key Share Inside

MtA/OLE dominates the cost ⇒ call it **once**, and keep it **away from the key**





prepared **before the key is known**
usable with **any key share**
each instance **used once**

Goal: one MtA/OLE whose inputs **never include a key share**

Key Idea: Align the Key Sharing with the Nonce Sharing

key, additive: $x = x_1 + x_2$

P_1 holds x_1, k_1, r_1

nonce, linear: $k = k_1 (r_1 + k_2)$

P_2 holds x_2, k_2

Key Idea: Align the Key Sharing with the Nonce Sharing

key, additive: $x = x_1 + x_2$

nonce, linear: $k = k_1 \, (r_1 + k_2)$

P_1 holds x_1, k_1, r_1 P_2 holds x_2, k_2

1. one MtA/OLE: P_1 inputs a fresh $x_1' \xleftarrow{\$} \mathbb{Z}_q$, P_2 inputs its nonce share k_2

   $t_A + t_B = x_1' \, k_2$

   neither input is a key share

Key Idea: Align the Key Sharing with the Nonce Sharing

key, additive: $x = x_1 + x_2$

nonce, linear: $k = k_1 \, (r_1 + k_2)$

P_1 holds x_1, k_1, r_1 P_2 holds x_2, k_2

1. one MtA/OLE: P_1 inputs a fresh $x_1' \xleftarrow{\$} \mathbb{Z}_q$, P_2 inputs its nonce share k_2

   $t_A + t_B = x_1' \, k_2$

   neither input is a key share

2. P_1 sends $cc = t_A + x_1' r_1 - x_1$
   P_2 sets $x_2' = x_2 - (t_B + cc)$

   $x = x_1' \, (r_1 + k_2) + x_2'$

   key re-shared, aligned with the nonce

The key share enters only **after** the MtA/OLE, inside the single masked value cc.

Key Idea: Align the Key Sharing with the Nonce Sharing

key, additive: $x = x_1 + x_2$

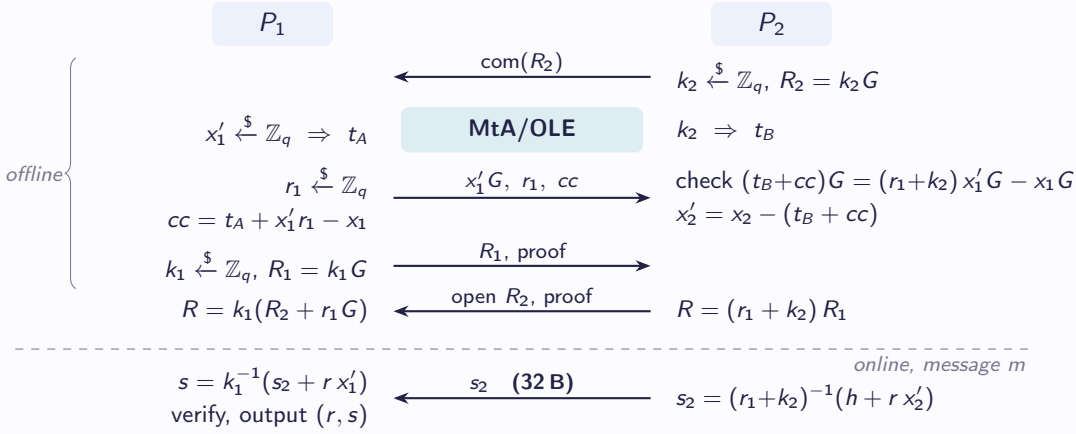
nonce, linear: $k = k_1\,(r_1 + k_2)$

P_1 holds x_1, k_1, r_1 P_2 holds x_2, k_2

1. one MtA/OLE: P_1 inputs a fresh $x_1' \xleftarrow{\$} \mathbb{Z}_q$, P_2 inputs its nonce share k_2 — $t_A + t_B = x_1'\,k_2$; neither input is a key share
2. P_1 sends $cc = t_A + x_1' r_1 - x_1$; P_2 sets $x_2' = x_2 - (t_B + cc)$ — $x = x_1'\,(r_1 + k_2) + x_2'$; key re-shared, aligned with the nonce
3. online, P_2 sends $s_2 = (r_1 + k_2)^{-1}(h + r\,x_2')$; P_1 outputs $s = k_1^{-1}(s_2 + r\,x_1')$ — $s = k^{-1}(h + r\,x)$; one field element online

The key share enters only **after** the MtA/OLE, inside the single masked value cc.

Menshen: Signing Flow

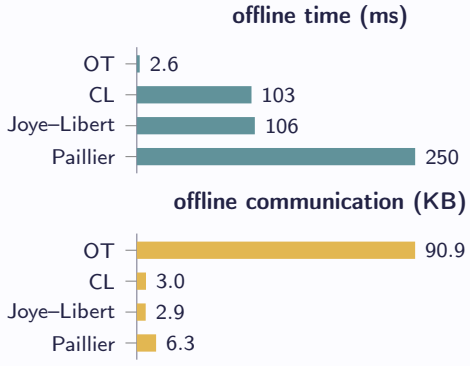


KeyGen among $n \geq 2$ (Shamir), any 2 sign · authenticated P2P, no broadcast
1 static malicious corruption · real/ideal, ROM, MtA/OLE as an ideal functionality
assumptions: those of the chosen MtA/OLE instantiation

[XAXYC21] com(·): hash commitment to R_2 and a proof of knowledge of k_2.

Menshen: Cost by MtA/OLE Instantiation

	offline		online
MtA/OLE	time	comm.	time, comm.
OT [DKLs]	2.6 ms	90.9 KB	**0.2 ms** **32 B**
CL	103 ms	3.0 KB	
Joye–Libert	106 ms	2.9 KB	
Paillier	250 ms	6.3 KB	



every instantiation: **one** MtA/OLE offline, **32 B** online

secp256k1, 128-bit security · Joye–Libert MtA/OLE: [XALCCXYZ23]
Intel Core i9-12900K with 64 GB of memory under Arch Linux

Octopus-CL: Threshold ECDSA with GOD in Three Rounds

Motivation: Guaranteed Output Delivery (GOD)

Identifiable abort (IA) any $t \leq n$

every message is publicly checkable, so cheaters are named — even against a **malicious majority**

Robustness $n \geq 2t - 1$

cheaters are excluded and the remaining $\geq t$ parties still output a signature — needs an **honest majority**

channels: authenticated P2P + **broadcast**, so that all honest parties agree on who cheated

protocol	type	rounds	IA	Robustness	online cost
DKLs24	OT	3	✗	✗	optimal
CDKS24	OT	7	✓	✗	2× DDH proof
CGGMP20	Paillier	4	✓	✗	Enc + Paillier proofs
WMY+23	LHE	5+2	✓	✓†	3× DDH proof
WMC24	threshold CL	4	✓	✓	Dec + CL proofs
Octopus-CL	CL	**3**	✓	✓	**one DDH proof**

+2: extra rounds for identifiable abort † insecure once combined with identifiable abort (Idea 2) [TX25]

Idea 1: Leave the Temporary Nonce γ Undetermined

$$s = \frac{\gamma\,(h + r\,x)}{\gamma\,k}$$

γ: temporary nonce, cancels

prior [WMY+23]: t-out-of-n share γ **before** the MtAs
$\Rightarrow$ one extra round

Octopus-CL: each P_i keeps a plain additive share γ_i;
$\gamma = \sum_{i \in S} \gamma_i$ is fixed only in the **last** round

γ never needs to be pinned down $\Rightarrow$ **no round** to threshold-share it

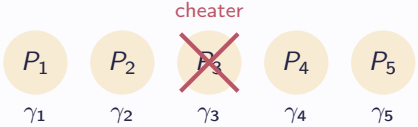
Idea 1: Leave the Temporary Nonce γ Undetermined

$$s = \frac{\gamma\,(h + r\,x)}{\gamma\,k}$$

γ: temporary nonce, cancels

prior [WMY+23]: t-out-of-n share γ **before** the MtAs
$\Rightarrow$ one extra round

Octopus-CL: each P_i keeps a plain additive share γ_i;
$\gamma = \sum_{i \in S} \gamma_i$ is fixed only in the **last** round



P_3 excluded $\Rightarrow \gamma' = \gamma - \gamma_3$

$s = \dfrac{\gamma'(h + r\,x)}{\gamma'\,k}$ still cancels

γ never needs to be pinned down $\Rightarrow$ **no round** to threshold-share it

Idea 2: GOD Needs Public Checking

private checking [WMY+23]: only Alice can check Bob, with her secret output α

1. adversary holds $t - 1$ parties; its Bob deviates in one MtA

Idea 2: GOD Needs Public Checking

private checking [WMY+23]: only Alice can check Bob, with her secret output α

1. adversary holds $t - 1$ parties; its Bob deviates in one MtA
2. to name Bob, honest Alice reveals her output α

Idea 2: GOD Needs Public Checking

private checking [WMY+23]: only Alice can check Bob, with her secret output α

1. adversary holds $t-1$ parties; its Bob deviates in one MtA
2. to name Bob, honest Alice reveals her output α
3. adversary solves $a = b^{-1}(\alpha + \beta) =$ Alice's nonce share

Idea 2: GOD Needs Public Checking

private checking [WMY+23]: only Alice can check Bob, with her secret output α

1. adversary holds $t-1$ parties; its Bob deviates in one MtA
2. to name Bob, honest Alice reveals her output α
3. adversary solves $a = b^{-1}(\alpha + \beta) =$ Alice's nonce share
4. plus its own $t-1$ shares: the nonce k

Idea 2: GOD Needs Public Checking

private checking [WMY+23]: only Alice can check Bob, with her secret output α

1. adversary holds $t-1$ parties; its Bob deviates in one MtA
2. to name Bob, honest Alice reveals her output α
3. adversary solves $a = b^{-1}(\alpha + \beta)$ = Alice's nonce share
4. plus its own $t-1$ shares: the nonce k
5. robustness: signing still outputs (r, s)
   $\Rightarrow$ signing key $x = r^{-1}(k\,s - h)$

Idea 2: GOD Needs Public Checking

private checking [WMY+23]: only Alice can check Bob, with her secret output α

1. adversary holds $t-1$ parties; its Bob deviates in one MtA
2. to name Bob, honest Alice reveals her output α
3. adversary solves $a = b^{-1}(\alpha + \beta)$ = Alice's nonce share
4. plus its own $t-1$ shares: the nonce k
5. robustness: signing still outputs (r, s)
   $\Rightarrow$ signing key $x = r^{-1}(k\,s - h)$

IA + robustness + private checking
$\Rightarrow$ **key recovery**

Idea 2: GOD Needs Public Checking

private checking [WMY+23]: only Alice can check Bob, with her secret output α

1. adversary holds $t-1$ parties; its Bob deviates in one MtA
2. to name Bob, honest Alice reveals her output α
3. adversary solves $a = b^{-1}(\alpha + \beta)$ = Alice's nonce share
4. plus its own $t-1$ shares: the nonce k
5. robustness: signing still outputs (r, s)
   $\Rightarrow$ signing key $x = r^{-1}(k\,s - h)$

IA + robustness + private checking

$\Rightarrow$ **key recovery**

our multi-party MtA

public checking: CL ciphertexts + proofs anyone can verify

cheaters excluded, **no complaint or dispute phase**

per-pair proofs batched into **one constant-size proof**

Octopus-CL in Three Rounds

Round 1 offline

- $\gamma_i \xleftarrow{\$} \mathbb{Z}_q$: $\mathrm{Enc}(\gamma_i)$ + proof (first MtA message)
- publicly verifiable shares of a fresh nonce k (distributed randomness generation)

Round 2 offline

- combine: t-out-of-n share k_i, $R_i = k_i G$
- second MtA messages on k_i and x_i, one batched proof each
- keep shares of $k\gamma$ and $x\gamma$; $R = \sum_i \lambda_i R_i$, r

Online $h = H(m)$

- masked shares of $k\gamma$ and $(h + r\,x)\gamma$
- DDH proof: the same γ_i in $\gamma_i R$ and $\gamma_i(hG + rX)$
- $s = \dfrac{(h + r\,x)\gamma}{k\gamma}$, verify

Idea 3: online consistency of γ

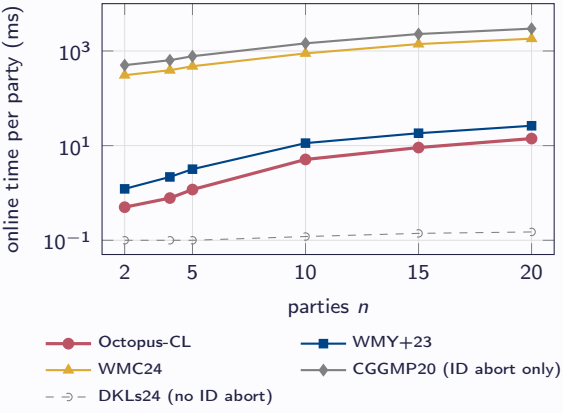
no public γG exists; the DDH relation between $\gamma_i R$ and $\gamma_i(hG + rX)$ is proved over the curve: **a few EC operations**, no Paillier/CL proofs

signature does not verify?

rebuild each $\gamma_j R$ and $\gamma_j(hG + rX)$ from P_j's shares and public values, check its DDH proof, drop cheaters, re-assemble: **no extra round**

UC-realizes $\mathcal{F}_{\mathrm{ECDSA}}$ · static, up to $t-1$ corruptions · HSM + smoothness of CL · ID abort for any t, robustness for $n \geq 2t-1$

Octopus-CL: Performance



offline, per party

	$n = 2$	$n = 5$
WMY+23	506 ms, 2.9 KB	1326 ms, 6.2 KB
WMC24	1023 ms, 4.6 KB	1584 ms, 4.6 KB
Octopus-CL	**406 ms, 2.3 KB**	**1224 ms, 4.9 KB**

online at $n = 2$: 0.5 ms vs. 308 ms (WMC24)

caveat: WMC24 sends $O(1)$ and computes $O(n)$, so our *total* cost is lower only for $n \leq 12$ (time) and $n \leq 5$ (bandwidth) — NIST's two-, three- and small-party profiles

[TX25]: C++/BICYCL, Apple M1 Pro, secp256k1, $\lambda = 128$; per-party running time and sending communication.

Octopus-TCL: Constant-Size Messages in Three Rounds

Motivation: Constant-Size Messages, in Three Rounds

	3 rounds	≥ 4 rounds
message size $O(n)$	DKLs24 OT	CCL+23 (7) WMY+23 (6)
message size $O(1)$	**Octopus-TCL** this work	WMC24 (4) threshold CL

threshold CL encryption [BDO23]

message size **independent of** n, unlike pairwise MtAs

why WMC24 needs 4 rounds

it signs $s = k(h + r\,x)$ with $R = k^{-1}G = (k\varphi)^{-1}(\varphi G)$, available only after a threshold decryption

$\Rightarrow$ r after round 3, s in round 4

Goal: the **constant message size** of threshold ECDSA, in **3 rounds** instead of 4

[JTX25, Table 1]

Octopus-TCL: Keep ECDSA in Its Original Form

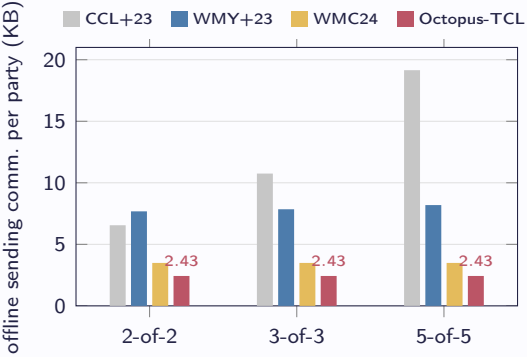
	round 1	round 2	round 3	round 4
WMC24	Enc(k_i), ElGamal($\varphi_i G$)	Enc($k\,x_i$), Enc($k\varphi_i$)	threshold-decrypt $k\varphi$ and φG $\Rightarrow$ R, r	*online*: threshold-decrypt Enc($k(h+rx)$) $\Rightarrow$ s
Octopus-TCL	Enc(φ_i) under the threshold-CL key; commit to $R_i = k_i G$	Enc(φx_i), Enc(φk_i) + proofs; open R_i $\Rightarrow$ R, r	*online*: threshold-decrypt Enc(φk) and Enc($\varphi(h+rx)$)	**one round saved**

keep $s = k^{-1}(h + r\,x)$:
R and r are **fixed in the offline rounds**

Enc($\varphi(h + rx)$) built locally;
$s = \varphi(h + r\,x) \,/\, \varphi k$: φ cancels

authenticated P2P, no broadcast · up to $t-1$ corruptions · security with abort · real/ideal, ROM · IND-CPA of threshold CL, Two online CL Dec

Octopus-TCL: Cost



total time per party (ms)

	2-of-2	3-of-3	5-of-5
CCL+23	1012	1518	2531
WMY+23	1730	2595	4328
WMC24	631	938	1564
Octopus-TCL	**540**	**810**	**1350**

2426 B per party in every setting: the message size does not grow with n

[JTX25]: C++/BICYCL, Ryzen 7 PRO 4750U VM, secp256k1, 1827-bit Δ_K.

Summary

Takeaways

Menshen

one MtA/OLE, no key share inside

prepared before the key is known

32 B online

Octopus-CL

GOD in 3 rounds

ID abort for any t, robust for $n \geq 2t - 1$

EC-only online

Octopus-TCL

constant-size messages in 3 rounds

from threshold CL, independent of n

no broadcast



Contact: octopus_mptc@googlegroups.com

SoK and Open-Source Library

SoK: Threshold ECDSA from Theory to Practice
B. Jiang, G. Tang, C. Zhang, H. Xue · **IEEE S&P 2027**
Available at ePrint soon

tecdsa-lib: Rust library for threshold and two-party ECDSA, built to support the NIST Threshold Call

- 13 multi-party + 4 two-party protocols in one framework
- includes Menshen (XAL21), Octopus-CL (TX25), Octopus-TCL (JTX25)
- MIT OR Apache-2.0

multi-party: GGN16, GG18, LN18, CGGMP20, DKLs23, XAL23, WMY23, TX25, JTX25, WMC24, LLZ25, DNP25, KU25
two-party: Lin17, KGG24, XAL21, ABC24

github.com/
Jiangjiang-jiang/
tecdsa-lib

Thank you!

Questions?

octopus_mptc@googlegroups.com

SoK and Open-Source Library

SoK: Threshold ECDSA from Theory to Practice
B. Jiang, G. Tang, C. Zhang, H. Xue · **IEEE S&P 2027**
Available at ePrint soon

tecdsa-lib: Rust library for threshold and two-party ECDSA, built to support the NIST Threshold Call

- 13 multi-party + 4 two-party protocols in one framework
- includes Menshen (XAL21), Octopus-CL (TX25), Octopus-TCL (JTX25)
- MIT OR Apache-2.0

multi-party: GGN16, GG18, LN18, CGGMP20, DKLs23, XAL23, WMY23, TX25, JTX25, WMC24, LLZ25, DNP25, KU25
two-party: Lin17, KGG24, XAL21, ABC24

Guofeng Tang
is on the job market

postdoc at SMU

threshold cryptography,
PQC, ZK proofs, MPC

github.com/
Jiangjiang-jiang/
tecdsa-lib

Thank you!

Questions?

octopus_mptc@googlegroups.com

victoriatang.
github.io

References

[XAXYC21] Xue, Au, Xie, Yuen, Cui. Efficient online-friendly two-party ECDSA signature. ACM CCS 2021. (Menshen; ePrint 2022/318)

[TX25] Tang, Xue. Robust threshold ECDSA with online-friendly design in three rounds. IEEE S&P 2025. (Octopus-CL; ePrint 2025/910)

[JTX25] Jiang, Tang, Xue. Three-round (robust) threshold ECDSA from threshold CL encryption. ACISP 2025. (Octopus-TCL; ePrint 2026/190)

[JTZX27] Jiang, Tang, Zhang, Xue. SoK: Threshold ECDSA from theory to practice. IEEE S&P 2027.

[BDO23] Braun, Damgård, Orlandi. Secure multiparty computation from threshold encryption based on class groups. CRYPTO 2023.

[CCL+23] Castagnos, Catalano, Laguillaumie, Savasta, Tucker. Bandwidth-efficient threshold EC-DSA revisited: online/offline extensions, identifiable aborts, proactive and adaptive security. TCS 2023.

[CD24] Cascudo, David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. EUROCRYPT 2024.

[CDKS24] Cohen, Doerner, Kondi, shelat. Secure multiparty computation with identifiable abort from vindicating release. ePrint 2023/1136.

[CGGMP20] Canetti, Gennaro, Goldfeder, Makriyannis, Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. ACM CCS 2020.

[CL15] Castagnos, Laguillaumie. Linearly homomorphic encryption from DDH. CT-RSA 2015.

[DKLs24] Doerner, Kondi, Lee, shelat. Threshold ECDSA in three rounds. IEEE S&P 2024.

[WMY+23] Wong, Ma, Yin, Chow. Real threshold ECDSA. NDSS 2023.

[WMC24] Wong, Ma, Chow. Secure multiparty computation of threshold signatures made more efficient. NDSS 2024.

[XALCCXYZ23] Xue, Au, Liu, Chan, Cui, Xie, Yuen, Zhang. Efficient multiplicative-to-additive function from Joye–Libert cryptosystem and its application to threshold ECDSA. ACM CCS 2023.

[FIPS 186-5] NIST. Digital Signature Standard (DSS). 2023.

[NIST IR 8214C] Brandão, Peralta. NIST first call for multi-party threshold schemes. 2026.