Compact Zero-Knowledge Proofs for Threshold ECDSA with Trustless Setup

Tsz Hon Yuen [1] Handong Cui [1] Xiang Xie [2]

[1] The University of Hong Kong [2] MatrixElements Technologies

PKC 2021

May 10, 2021

Threshold signature allows n parties to share the message signing ability without trusting each other, such that no coalition of $t < n$ or fewer users can generate a valid signature.

Threshold ECDSA in Practice

A threshold signature with $t = 1, n = 3$ is useful for a hot wallet of a crypto exchange

- the exchange holds sk_1 for online transaction and sk_2 for paper backup, and a separate security firm holds sk_3 to validate transactions
- losing one key from the exchange or the security firm does not compromise the hot wallet.

Most blockchain systems just trivially check if $t + 1$ signatures are valid...

- Improve the efficiency of ZK proofs used in two-party and threshold ECDSA.
- When applied to two-party ECDSA: the bandwidth of KeyGen $\Downarrow$ 47%, and the running time for KeyGen and Sign $\Uparrow$ 35% and $\Uparrow$ 104% faster respectively.
- When applied to threshold ECDSA:
  - Scheme 1: optimized for KeyGen (about $\Downarrow$ 70% bandwidth and $\Uparrow$ 85% faster computation in KeyGen , at a cost of 20% larger bandwidth in Sign)
  - Scheme 2: all-rounded performance improvement (about $\Downarrow$ 60% bandwidth, $\Uparrow$ 46% faster computation in KeyGen without additional cost in Sign).

Many ZK proofs are involved in threshold ECDSA.

We improve the existing ZK proofs involved in two-party/threshold ECDSA.

Table: Modifications to the threshold ECDSA in [2] are shown in the box.

P_i	$\mathsf{IKeyGen}(\mathrm{param})$	All players $\{P_j\}_{j \neq i}$
$u_i \xleftarrow{\$} \mathbb{Z}_q$		
$(\mathsf{kgc}_i, \mathsf{kgd}_i) \leftarrow \mathsf{Com}(\hat{P}^{u_i})$		
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}()$	$\xrightarrow{\mathsf{kgc}_i, \mathsf{pk}_i}$	
	$\xrightarrow{\mathsf{kgd}_i}$	
$\pi_k := \mathsf{ZKPoKRepS}(\mathsf{pk}_i; \mathsf{sk}_i : \mathsf{pk}_i = g_q^{\mathsf{sk}_i})$	$\xleftrightarrow{\pi_k}$	Abort if the proof fails.
Follow from line 5 of Fig. 4 in in [2].		

P_i	$\mathsf{ISign}(\mathrm{param}, m)$ Phase 1	All players $\{P_j\}_{j \neq i}$
$k_i, \gamma_i \xleftarrow{\$} \mathbb{Z}_q,\ r_i \xleftarrow{\$} [0, S]$		
$(\mathsf{c}_i, \mathsf{d}_i) \leftarrow \mathsf{Com}(\hat{P}^{\gamma_i})$		
$C_{k_i} \leftarrow \mathsf{CL.Enc}(\mathsf{pk}_i, k_i; r_i)$	$\xrightarrow{C_{k_i}, \mathsf{c}_i}$	
$\pi_C := \mathsf{ZKPoKEnc}((k_i, r_i) :$		
$((\mathsf{pk}_i, C_{k_i}); (k_i, r_i)) \in \mathcal{R}_{\mathsf{Enc}})$	$\xleftrightarrow{\pi_C}$	Abort if the proof fails.

Most two-party or threshold ECDSA schemes use additive homomorphic encryption.

- Some earlier papers [5, 6, 4] used Paillier encryption.
- [1] used the additive homomorphic Castagnos-Laguillaumie (CL) encryption [3].

Additive Homomorphic CL Encryption

- based on an unknown order group G, which contains a subgroup F in which the DL problem is tractable.
- hard subgroup membership (HSM) assumption holds in G
- can be constructed from class groups of quadratic fields.

CL vs. Paillier encryption: the generation of the class group is trustless, and $|\text{class group element}| < |\text{Paillier group element}|$.

- The group G^q is a group of unknown order s with a genertor g_q.
- F is a group of known order q with a genertor f.
- By construction $G = G^q \times F$ and $g := f \cdot g_q$ is the generator of G.
- The DL problem in F can be solved by a polynomial time algorithm Solve:

  $$x \leftarrow \mathsf{Solve}(f^x), \quad \forall x \xleftarrow{\$} \mathbb{Z}_q.$$

  For simplicity, we will call this group the HSM group.
- The HSM group can be instantiated by class groups of imaginary quadratic order.

CL Encryption:

- **Setup**. On input a security parameter 1^λ and a prime q, it runs $\mathcal{G}_{\mathrm{HSM}} \leftarrow \mathsf{GGen}_{\mathrm{HSM},q}(1^\lambda)$. It parses $\mathcal{G}_{\mathrm{HSM}} = (\tilde{s}, g, f, g_q, \tilde{G}, G, F, G^q)$. Define $S = \tilde{s} \cdot 2^{\epsilon_d}$ for some statistical distance ϵ_d. It outputs $\mathsf{param} = \mathcal{G}_{\mathrm{HSM}}$. The input param is omitted for other algorithms for simplicity.
- **KeyGen**. It picks a random $\mathsf{sk} \xleftarrow{\$} [0, S]$ and computes $\mathsf{pk} = g_q^{\mathsf{sk}}$. It returns $(\mathsf{sk}, \mathsf{pk})$.
- **Encrypt**. On input a public key pk and a message m, it picks a random $\rho \xleftarrow{\$} [0, S]$ and outputs the ciphertext $C = (C_1, C_2)$, where:

$$C_1 = f^m \mathsf{pk}^\rho, \quad C_2 = g_q^\rho.$$

- **Decrypt**. On input a secret key sk and a ciphertext $C = (C_1, C_2)$, it computes $M = C_1 / C_2^{\mathsf{sk}}$ and returns $m \leftarrow \mathsf{Solve}(M)$.

CL Encryption (cont.):

- EvalScal. On input a public key pk, a ciphertext $C = (C_1, C_2)$ and a scalar s, it outputs $C' = (C'_1 = C_1^s, C'_2 = C_2^s)$.
- EvalSum. On input a public key pk, two ciphertexts $C = (C_1, C_2)$ and $C' = (C'_1, C'_2)$, it outputs $\hat{C} = (\hat{C}_1 = C_1C'_1, \hat{C}_2 = C_2C'_2)$.

Security based on the the hard subgroup membership (HSM) assumption: hard to distinguish the elements of G^q in G.

Technical difficulties: Efficient ZK proofs in the HSM group for

1. DL of an unknown order group element ($\mathsf{pk} = g_q^{\mathsf{sk}}$)
2. well-formedness of a CL ciphertext ($C_1 = f^m \mathsf{pk}^\rho, C_2 = g_q^\rho$)

Existing Works

- [1] used a ZK proof with a single bit challenge. To achieve soundness error of $2^{-\epsilon_s}$, the protocol has to be repeated for ϵ_s-times $\rightarrow$ inefficient.
- [2] tackled the first DL problem by using a lowest common multiple (lcm) tricks, which reduces the repetition of the ZK proof to about $\epsilon_s/10$-times. [2] tackled the second problem based on a strong root assumption in the HSM group.

1. ZK proof for DL in HSM group in [2] only reduces the repetition by 10 times (e.g. from 80 times to 8 times for soundness error 2^{80}.)
2. ZK proof for CL ciphertext in [2] does not allow a fast, trustless setup.

- [2] use the strong root assumption that when given a random group element $w \in G \setminus F$, it is difficult to output a group element u and a positive integer $e \neq 2^k$ such that $u^e = w$.
- However, a random group generator w can only be obtained from:
  - a standardized group: all users have to trust the standardizing authority → not desirable for decentralized applications such as public blockchain.
  - jointly generated by all participating parties during the interactive KeyGen → greatly increases the round complexity and the bandwidth used.

We first consider a ZK proof for a simple DL relation $\mathcal{R}$ in an unknown order group G for some group elements $g, w \in G \setminus F$:[1]

$$\mathcal{R} = \{x \in \mathbb{Z} : w = g^x\}.$$

The subgroup F makes the ZK proof on the relation $\mathcal{R}$ much more complicated.

[1] Since it is easy to compute $\log_g w$ if $g \in F$, it is impossible to construct a ZK proof for $\mathcal{R}$ if $g \in F$. Hence, we restrict that $g \in G \setminus F$.

Attempt 1: Use adaptive root assumption

- The adversary first selects a group element $w \in G \setminus F$. Given a random prime ℓ, no PPT adversary can output a group element u such that $u^{\ell} = w$.

Algorithm 1: Insecure ZK Proof for the relation $\mathcal{R}$

1. Verifier sends a random λ-bit prime ℓ.
2. Prover finds $q' \in \mathbb{Z}$ and $r \in [0, \ell - 1]$ s.t. $x = q'\ell + r$. Prover sends $Q = g^{q'}$ and r to the verifier.
3. Verifier accepts if $r \in [0, \ell - 1]$ and $Q^{\ell} g^{r} = w$.

It is insecure:

- If the prover knows x and y such that $w = g^x f^y$ for some $f \in F$, he can compute $Q' = g^{q'} f^{\frac{y}{\ell}}$ since the order of f is known. It can pass the verification since:

$$Q'^{\ell} g^r = (g^{q'} f^{\frac{y}{\ell}})^{\ell} g^r = g^x f^y = w.$$

ZK Proof DL for HSM Group

Our solution: use an extra round of challenge to eliminate the elements of order q in w.

This extra round simply uses q instead of using the prime number ℓ.

Algorithm 2: ZK Proof for the relation $\mathcal{R}$

Param: A security parameter B.

1. Prover chooses $k \xleftarrow{\$} [-B, B]$ and sends $R = g^k$ to the verifier.
2. Verifier sends $c \xleftarrow{\$} [0, q-1]$ to the prover.
3. Prover computes $s = k + cx$. Prover finds $d \in \mathbb{Z}$, $e \in [0, q-1]$ s.t. $s = dq + e$ and sends $D = g^d$ and e to the verifier.
4. If $e \in [0, q-1]$ and $D^q g^e = Rw^c$, verifier sends a random λ-bit prime ℓ.
5. Prover finds $q' \in \mathbb{Z}$ and $r \in [0, \ell-1]$ s.t. $s = q'\ell + r$. Prover sends $Q = g^{q'}$ and r to the verifier.
6. Verifier accepts if $r \in [0, \ell-1]$ and $Q^\ell g^r = Rw^c$.

Safe against previous attack:

- If the prover knows x and y such that $w = g^x f^y$ for some $f \in F$, and if he can pass the verification:

$$D^q g^e = Rw^c = R(g^x f^y)^c$$

RHS has no element in F $\rightarrow$ f^{cy} is cancelled out by R $\rightarrow$ negligible probability since c is given after R.

Our protocol only runs for one time only for a soundness error of $2^{-\epsilon_s}$, as compared to ϵ_s-times for [1] and $\epsilon_s/10$-times for [2].

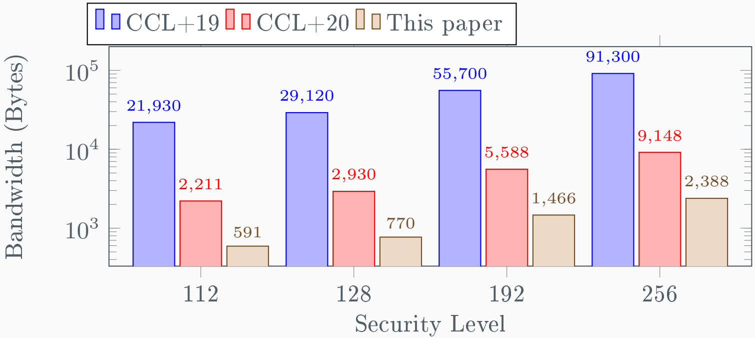


Figure: Comparison of ZK Proof of DL relation in HSM group.

97% shorter than CCL+19 [1] and around 74% shorter than CCL+20 ([2], §5.1) with the same level of soundness error and statistical distance of 2^{-80}.

- As compared with ZK proofs in [2], their strong root assumption is similar to the strong RSA assumption, while our adaptive root assumption is more similar to the RSA assumption.
- [2] gives a ZK proof for a modified relation: $h^y = g^x$ for some public value y.
- The security of our ZK proofs requires the use of generic group model while the security of the ZK proofs in [2] does not.

ZK Proof for CL ciphertext:

$$\mathcal{R}_{\mathsf{Enc}} = \{(\mathsf{pk}, \mathrm{C}_1, \mathrm{C}_2); (\mathrm{m}, \rho) | \mathsf{pk} \in \mathrm{G}^{\mathrm{q}}, \rho \in [0, \mathrm{S}] : \mathrm{C}_1 = \mathrm{f}^{\mathrm{m}} \mathsf{pk}^{\rho} \wedge \mathrm{C}_2 = \mathrm{g}_{\mathrm{q}}^{\rho}\}.$$

Similar to the DL proof, but special care is needed for the term m, since the order of f is known.

Algorithm 5: Protocol ZKPoKEnc for the relation $\mathcal{R}_{\mathsf{Enc}}$

Param: $\mathcal{G}_{\mathrm{HSM}} \leftarrow \mathsf{GGen}_{\mathrm{HSM},q}(1^\lambda)$, $B = 2^{\lambda+\epsilon_d+2}\tilde{s}$, where $\epsilon_d = 80$.

Input: $C_1, C_2, \mathsf{pk} \in G^q$.

Witness: $\rho \in [0, S], m \in \mathbb{Z}_q$, where $S = \tilde{s} \cdot 2^{\epsilon_d}$.

1 Prover chooses $s_\rho \xleftarrow{\$} [-B, B]$, $s_m \xleftarrow{\$} \mathbb{Z}_q$ and computes:

$$S_1 = \mathsf{pk}^{s_\rho} f^{s_m}, \quad S_2 = g_q^{s_\rho}.$$

Prover sends (S_1, S_2) to the verifier.

2 Verifier sends $c \xleftarrow{\$} [0, q-1]$ to the prover.

3 Prover computes:

$$u_\rho = s_\rho + c\rho, \quad u_m = s_m + cm \mod q.$$

Prover finds $d_\rho \in \mathbb{Z}$ and $e_\rho \in [0, q-1]$ s.t. $u_\rho = d_\rho q + e_\rho$. Prover computes:

$$D_1 = \mathsf{pk}^{d_\rho}, \quad D_2 = g_q^{d_\rho}.$$

Prover sends (u_m, D_1, D_2, e_ρ) to the verifier.

4 The verifier checks if $e_\rho \in [0, q-1]$ and:

$$D_1^q \mathsf{pk}^{e_\rho} f^{u_m} = S_1 C_1^c, \quad D_2^q g_q^{e_\rho} = S_2 C_2^c.$$

If so, the verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.

5 Prover finds $q_\rho \in \mathbb{Z}$ and $r_\rho \in [0, \ell-1]$ s.t. $u_\rho = q_\rho \ell + r_\rho$. Prover computes:

$$Q_1 = \mathsf{pk}^{q_\rho}, \quad Q_2 = g_q^{q_\rho}.$$

Prover sends (Q_1, Q_2, r_ρ) to the verifier.

6 Verifier accepts if $r_\rho \in [0, \ell-1]$ and:

$$Q_1^\ell \mathsf{pk}^{r_\rho} f^{u_m} = S_1 C_1^c, \quad Q_2^\ell g_q^{r_\rho} = S_2 C_2^c.$$

Table: Comparison of communication size for ZK proof of the well-formedness of CL ciphertext.

	Communication Size (Bytes)				Requirement
	$\lambda = 112$	$\lambda = 128$	$\lambda = 192$	$\lambda = 256$	
CCL+19 [1]	37970	49950	95520	156130	$\times$
CCL+20 [2]	495	645	1214	1972	Random $g_q \in G^q$
This paper	1129	1488	2864	4692	$\mathsf{pk} \in G^q$, GGM

- Note that CCL+20 [2] required that g_q is randomly chosen in G^q prior to running the ZK proof $\rightarrow$ g_q jointly generated by all participating parties $\rightarrow$ overheads in bandwidth as well as a few more rounds of communication.
- Our scheme additionally require that $\mathsf{pk} \in G^q$. It can be proved by the owner of the secret key separately (to be used in threshold ECDSA), or can be embedded into this ZK proof if the prover himself is also the owner of the secret key (to be used in 2-party ECDSA).

We mainly use the two-party ECDSA protocols in [1].

For the ZK proof part, we have to prove the relation:

$$\mathcal{R}_{\mathsf{Enc}'} = \{(\mathrm{m}, \rho, \mathsf{sk}) : \mathrm{C}_1 = \mathrm{f}^{\mathrm{m}} \mathsf{pk}^{\rho} \wedge \mathrm{C}_2 = \mathrm{g}_{\mathrm{q}}^{\rho} \wedge \hat{\mathrm{Q}} = \hat{\mathrm{P}}^{\mathrm{m}} \wedge \mathsf{pk} = \mathrm{g}_{\mathrm{q}}^{\mathsf{sk}}\}.$$

Table: Comparison for two-party ECDSA with different security levels.

	Security Level	IKeyGen (Bytes)	ISign (Bytes)	Assumption
CCL+19 [1]	$\lambda = 112$	38714	575	Hard subgroup membership
	$\lambda = 128$	50876	697	
	$\lambda = 192$	97230	1260	
	$\lambda = 256$	158850	1973	
CCL+19-lcm [2]	$\lambda = 112$	4559	575	Hard subgroup membership
	$\lambda = 128$	5939	697	
	$\lambda = 192$	11280	1260	
	$\lambda = 256$	18351	1973	
Our two-party ECDSA	$\lambda = 112$	2453	575	Hard subgroup membership, adaptive root subgroup.
	$\lambda = 128$	3173	697	
	$\lambda = 192$	6030	1260	
	$\lambda = 256$	9789	1973	

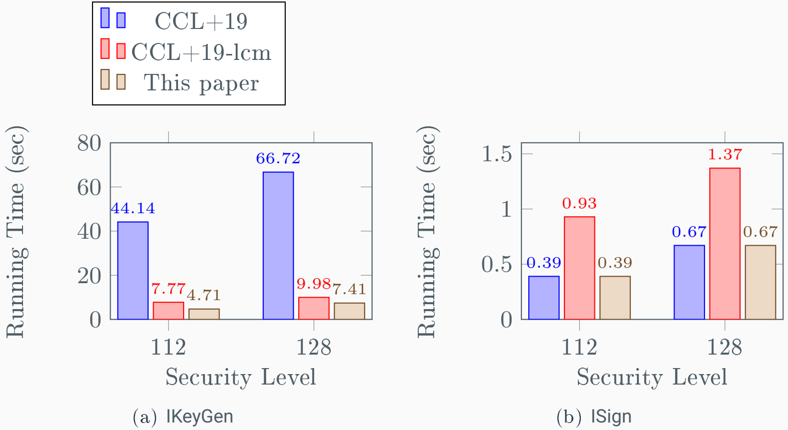


Figure: Running time of two-party ECDSA.

Table: Scheme 1: Modifications to the threshold ECDSA in [2] are shown in the box.

	IKeyGen(param)	
P_i		All players $\{P_j\}_{j \neq i}$
$u_i \xleftarrow{\$} \mathbb{Z}_q$		
$(\mathsf{kgc}_i, \mathsf{kgd}_i) \leftarrow \mathsf{Com}(\hat{P}^{u_i})$		
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}()$	$\xrightarrow{\mathsf{kgc}_i, \mathsf{pk}_i}$	
	$\xrightarrow{\mathsf{kgd}_i}$	
$\pi_k := \mathsf{ZKPoKRepS}(\mathsf{pk}_i; \mathsf{sk}_i : \mathsf{pk}_i = g_q^{\mathsf{sk}_i})$	$\xleftrightarrow{\pi_k}$	Abort if the proof fails.
Follow from line 5 of Fig. 4 in in [2].		

	ISign(param, m) Phase 1	
P_i		All players $\{P_j\}_{j \neq i}$
$k_i, \gamma_i \xleftarrow{\$} \mathbb{Z}_q,\ r_i \xleftarrow{\$} [0, S]$		
$(\mathsf{c}_i, \mathsf{d}_i) \leftarrow \mathsf{Com}(\hat{P}^{\gamma_i})$		
$C_{k_i} \leftarrow \mathsf{CL.Enc}(\mathsf{pk}_i, k_i; r_i)$	$\xrightarrow{C_{k_i}, \mathsf{c}_i}$	
$\pi_C := \mathsf{ZKPoKEnc}((k_i, r_i) :$		
$((\mathsf{pk}_i, C_{k_i}); (k_i, r_i)) \in \mathcal{R}_{\mathsf{Enc}})$	$\xleftrightarrow{\pi_C}$	Abort if the proof fails.

If we make the extra adaptive root subgroup assumption, we can keep the ISign algorithm and the most of the IKeyGen algorithm in CCL+20 [2].

We only need to modify the interactive ISetup algorithm in [2], such that the proof of knowledge of t_i for $g_i = g_q^{t_i}$ is replaced by our ZKPoKRepS protocol.

Table 5: Comparison for threshold ECDSA with different security levels.

	Security Level	IKeyGen (Bytes)	ISign (Bytes)	Assumption
CCL+20 [5]	$\lambda = 112$	$32tn + 2692n - 64$	$2397t - 1412$	Hard subgroup membership Strong root subgroup
	$\lambda = 128$	$32tn + 3479n - 64$	$3100t - 1891$	
	$\lambda = 192$	$32tn + 6518n - 96$	$5862t - 3694$	
	$\lambda = 256$	$32tn + 10535n - 128$	$9489t - 6099$	
Our threshold ECDSA scheme 1	$\lambda = 112$	$32tn + 791n - 64$	$3031t - 1412$	Hard subgroup membership Adaptive root subgroup
	$\lambda = 128$	$32tn + 979n - 64$	$3944t - 1891$	
	$\lambda = 192$	$32tn + 1779n - 96$	$7512t - 3694$	
	$\lambda = 256$	$32tn + 2805n - 128$	$12210t - 6099$	
Our threshold ECDSA scheme 2	$\lambda = 112$	$32tn + 1072n - 64$	$2397t - 1412$	Hard subgroup membership Adaptive root subgroup Strong root subgroup
	$\lambda = 128$	$32tn + 1319n - 64$	$3100t - 1891$	
	$\lambda = 192$	$32tn + 2397n - 96$	$5862t - 3694$	
	$\lambda = 256$	$32tn + 3775n - 128$	$9489t - 6099$	

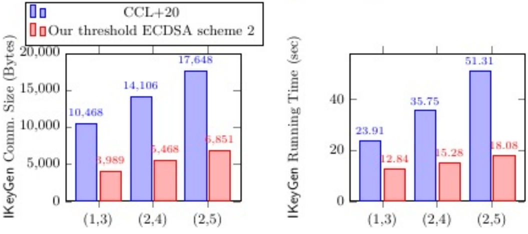
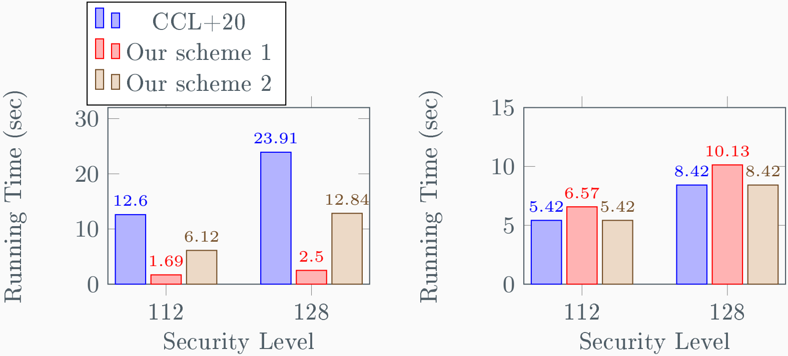


Fig. 2: (t, n)-Threshold ECDSA with 128-bit security.



(a) IKeyGen.

(b) ISign.

Figure: Running time of threshold ECDSA with $t = 1, n = 3$.

- We propose a compact zero-knowledge proof for the DL relation in HSM groups and the CL ciphertext.
- When applied to two-party ECDSA and threshold ECDSA, it can significantly improve the performance in terms of bandwidth used in IKeyGen, and the running time of IKeyGen and ISign.

[1] Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ECDSA from hash proof systems and efficient instantiations. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. Lecture Notes in Computer Science, vol. 11694, pp. 191–221. Springer (2019)

[2] Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-efficient threshold EC-DSA. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. Lecture Notes in Computer Science, vol. 12111, pp. 266–296. Springer (2020)

[3] Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from DDH. In: Nyberg, K. (ed.) CT-RSA 2015. Lecture Notes in Computer Science, vol. 9048, pp. 487–505. Springer (2015)

[4] Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) CCS 2018. pp. 1179–1194. ACM (2018)

[5] Lindell, Y.: Fast secure two-party ECDSA signing. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. Lecture Notes in Computer Science, vol. 10402, pp. 613–644. Springer (2017)

[6] Lindell, Y., Nof, A.: Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) CCS 2018. pp. 1837–1854. ACM (2018)