Distributed Key Generation Cryptography at Coinbase

Michael Lodder, Daniel Zhou

May 2021

1 Introduction

Distributed Key Generation (DKG) is a multi-party computation (MPC) process that outputs secret key shares among participants in such a way that no single party holds enough material to compute a full signature without the others. Coinbase supports three different signing algorithms ECDSA, EDDSA, and BLS. Each of these signing algorithms can be computed in this manner but have varying requirements. This paper discusses what these requirements are and how DKG is implemented to meet these requirements.

2 BLS Requirements

BLS signatures have the simplest requirement in that they only use the secret key shares and can compute a full signature in 1 round.

3 EDDSA Requirements

EDDSA signatures are similar if not the same as Schnorr signatures. These signatures also only use the secret key shares but require 2 rounds to compute a full signature [KG20].

4 ECDSA Requirements

ECDSA signatures were not designed to be completed in a distributed manner and are thus the most complex. In addition to secret key shares, they require help from other cryptography methods to complete the task. The simplest method is the 2-party (2P-ECDSA) model described by Lindell [Lin17]. This method requires using Paillier keys [Pai99] to utilize their homomorphic encryption properties when jointly computing the signature to ensure no single party cheats. However, using paillier keys requires creating a zero-knowledge proof (ZKP) that a Paillier public-key was generated correctly (ZK-PPK) and proof that the decryption of a given Paillier ciphertext is the discrete log of a given elliptic curve point (ZK-PDL). However, there has been recent work to use ElGamal encryption instead to improve performance [LNR18]. Doerner et. al. [DKLas18] also has a version that uses oblivious transfers and ElGamal.

To do more than the 2P-ECDSA model, the most efficient model is by Gennaro and Goldfeder [GG20] which adds the requirement to create a group of unknown order called proof parameters in order to compute the following proofs: proof of knowledge that a participant knows the discrete log of an elliptic-curve point (ZK-DL), and proof that a committed value is in some range of values.

Paillier keys and proof parameters are expensive to create because they mandate using two safe primes which take quite a bit of time to generate (around 3-10 seconds on modern hardware).

5 Summary of Requirements

- Generate secret shares verifiably (All)
- Generate paillier keys (ECDSA)
- Generate proof parameters (ECDSA)
- Generate ZKPs for paillier keys and proof parameters (ECDSA)

6 Key Generation Protocol

BLS and EDDSA do not require the expensive use of paillier keys and proof parameters meaning their DKG protocol can greatly simplified. However, we can simplify the ECDSA requirements also by looking at how the paillier keys and proof parameters are used. Initially, paillier keys and proof parameters must be created for each participant. Even so, they do not need to be mapped one-to-one for each signing key. They can be reused across multiple keys such that a participant has thousands of ECDSA keys and only one set of paillier keys and proof parameters. This allows ECDSA to use the same simple DKG protocol as BLS and EDDSA once the more expensive operations have been performed.

We use the DKG as described in [GJKR05] which uses both Feldman Verifiable Secret Sharing (VSS) [Fel87] and Pedersen VSS [Ped92]. This DKG can be completed in 3 rounds with only two network broadcasts to other participants. Since the DKG only uses elliptic curve cryptography (ECC), it can be computed quickly (in a hundreds of microseconds) per key. [GG20] only requires an extra round but no extra network broadcasts to complete. Further optimizations can be made if we separate paillier key generation from proof parameter generation. Each of those can be computed in two rounds and even rotated independently.

These algorithms are described in section 8.

7 Notation

Select a curve E that contains the following properties:

- k: The security parameter in bits. Note that k is an upper bound on the security level for the corresponding curve.
- p: The field modulus.
- q: The subgroup order.
- G: Points in the cyclic group of order p.
- P: The base point in G or G_1 if E is a pairing-friendly curve.
- 1_G: The point at infinity in G.
- $x \cdot P$: scalar multiplication with scalar x at point P.

Scalars operate in Z_q and are denoted as lower case letters.

Points operating in G are denoted as capital letters.

Other notations used by algorithms:

- $a||b$: byte concatenation of two elements a, b
- $\xleftarrow{\$} Z_{q*}$: a random non-zero value less than q.
- $\emptyset$: nil or null.
- p_i: integer id for participant at index i.
- $a \stackrel{?}{=} b$: test if a equals b.
- $\perp$: Abort

8 Algorithms

Algorithm 1 PedersenFeldmanShare

Input: $E, Q, x, t, \{p_1, ..., p_n\}$

- E: elliptic curve as describe in the section 7
- Q: a point in G. Q must not be equal to P.
- x: scalar in Z_{q*} with q from E that will be split into shares.
- t: positive integer, the threshold for recovering x.
- $\{p_1, ..., p_n\}$: the set of participant ids as positive integers

1. $r \xleftarrow{\$} Z_{q*}$
2. $\{a_1, ..., a_t\}$ = New Polynomial(x, t)
3. $\{b_1, ..., b_t\}$ = New Polynomial(r, t)
4. $\{x_1, ..., x_n\}$ = ShamirSplit$(x, \{a_1, ..., a_t\}, n)$
5. $\{r_1, ..., r_n\}$ = ShamirSplit$(r, \{b_1, ..., b_t\}, n)$
6. $\{X_1, ..., X_t\} = \{a_1 \cdot P + b_1 \cdot Q,\ \ldots\ , a_t \cdot P + b_t \cdot Q\}$
7. $\{R_1, ..., R_t\} = \{a_1 \cdot P,\ \ldots\ , a_t \cdot P\}$

Output: $\{X_1, ..., X_t\}, \{R_1, ..., R_t\}, \{x_1, ..., x_n\}, \{r_1, ..., r_n\}$

Algorithm 2 Gennaro DKG Round 1

Input: $E, Q, t, i, \{p_1, ..., p_n\}$

- E: elliptic curve as describe in the section 7
- Q: a point in G. Q must not be equal to P.
- t: positive integer, the threshold.
- i: positive integer for this participant.
- $\{p_1, ..., p_n\}$: the set of participant ids as positive integers.

1. $x_i \xleftarrow{\$} Z_{q*}$
2. $\{X_{i1}, ..., X_{it}\}, \{R_{i1}, ..., R_{it}\}, \{x_{i1}, ..., x_{in}\}, \{r_{i1}, ..., r_{in}\}$ = PedersenFeldmanShare$(E, Q, x_i, t, \{p_1, ..., p_n\})$
3. EchoBroadcast $\{X_{i1}, ..., X_{it}\}$ to all other participants.
4. P2PSend x_{ij}, r_{ij} to participant p_j in $\{p_1, ..., p_n\}_{i \neq j}$.

Output: $\{R_{i1}, ..., R_{it}\}, \{x_{i1}, ..., x_{in}\}, \{r_{i1}, ..., r_{in}\}$

Note: The Gennaro DKG described above is slightly adapted from the original paper [GJKR05]. In the original protocol, once a participant receives an invalid share, the participant will complain against the dealer (i.e., the participant sends the invalid share). Then the dealer is required to reveal the share for each complaining party P_i. If any of the revealed shares fails, the dealer is disqualified. Those disqualified parties are not allowed to contribute to the final secret the protocol will produce. In our Gennaro DKG protocol, if the participant receives invalid share from other parties, he will immediately abort and the protocol will reboot. This makes the protocol much simpler but the protocol may not be able to detect dishonest parties, that is, a dishonest party can keep sending invalid shares to other parties and let the protocol reboot purposely. Thus, if we apply it in our production, we need an alternative approach to detect dishonest parties during the process.

Algorithm 3 Gennaro DKG Round 2

Input: $E, Q, t, i, \{p_1, ..., p_n\}, \{x_{ji}\}_{j \neq i, j \in [n]}, \{r_{ji}\}_{j \neq i, j \in [n]}, \{X_{j1}, ..., X_{jt}\}_{j \in [n]}$

- E: elliptic curve as describe in the section 7
- Q: a point in G. Q must not be equal to P.
- t: positive integer, the threshold.
- i: positive integer for this participant.
- $\{p_1, ..., p_n\}$: the set of participant ids as positive integers.
- $\{x_{ji}\}_{i \neq j, j \in [n]}$: secret shares from other j participants to participant i p2psent from Algorithm 2. These values should be stored for further use.
- $\{r_{ji}\}_{i \neq j, j \in [n]}$: blinding shares from other j participants to participant i p2psent from Algorithm 2.
- $\{X_{ij}, ..., X_{it}\}_{i \neq j}$: the set of pedersen verifiers from other j participants to participant i broadcast from Algorithm 2.

1. $sk = x_i$
2. for j in $1, ..., n$
3. if $i = j$ continue
4. if PedersenVerify$(E, Q, x_{ji}, r_{ji}, \{X_{j1}, ..., X_{jt}\}) =$ false ; $\perp$
5. $sk = (sk + x_{ji}) \mod q$
6. EchoBroadcast $\{R_1, ..., R_t\}$ to all other participants.
7. Store sk as party i's secret key share.

Output: sk

Algorithm 4 Gennaro DKG Round 3

Input: $E, Q, t, i, \{p_1, ..., p_n\}, \{x_{ji}\}_{i \neq j, j \in [n]}, \{R_{j1}, ...R_{jt}\}_{i \neq j, j \in [n]}$

- E: elliptic curve as describe in the section 7
- Q: a point in G. Q must not be equal to P.
- t: positive integer, the threshold.
- i: positive integer for this participant.
- $\{p_1, ..., p_n\}$: the set of participant ids as positive integers.
- $\{x_{ji}\}_{i \neq j, j \in [n]}$: secret shares from other j participants to participant i broadcast from Algorithm 2 and stored in Algorithm 3.
- $\{R_{j1}, ..., R_{jt}\}_{i \neq j, j \in [n]}$: the set of feldman verifiers from other j participants broadcast from Algorithm 3.

1. Set $Pk = R_i$
2. for j in $1, ..., n$
3. if $i = j$ continue
4. if FeldmanVerify$(E, x_{ji}, \{R_{j1}, ..., R_{jt}\}) =$ false ; $\perp$
5. $Pk = Pk + R_{j1}$
6. Store Pk as the public verification key

Output: Pk, EchoBroadcast Success

Algorithm 5 Gennaro DKG Round 4

Only needed for tECDSA if using [GG20]

Input: $E, Pk, i, t, \{p_1, ..., p_n\}, \{R_{ij}, ...R_{it}\}_{i \neq j}, \{R_1, ...R_t\}$

- E: elliptic curve as describe in the section 7
- Pk: the public key computed in Algorithm 4.
- i: positive integer for this participant.
- t: positive integer, the threshold.
- $\{p_1, ..., p_n\}$: the set of participant ids as positive integers.
- $\{R_{ij}, ..., R_{it}\}_{i \neq j}$: the set of feldman verifiers from other j participants to participant i broadcast from Algorithm 3.
- $\{R_i, ..., R_t\}$: the feldman verifiers computed in Algorithm 2 for participant i.

1. $R = \{\{R_1, ..., R_t\}, \{R_{ij}, ..., R_{it}\}_{i \neq j}\}$
2. for j in $1, ..., n$
3. $\quad W_j = Pk$
4. $\quad$ for k in $1, ..., t$
5. $\quad\quad c_k = (p_j \cdot k) \mod q$
6. $\quad\quad W_j = Wj + c_k \cdot R_j$

Output: $\{W_1, ..., W_n\}$

Algorithm 6 Gennaro/Goldfeder Paillier DKG Round 1

Adapted from the [GG20] Pseudocode

Input: E, i, Pk

- E: elliptic curve as describe in the section 7
- i: positive integer for this participant.
- Pk: the public key computed in Algorithm 4.

1. $sk_i, pk_i =$ PaillierKeyGen()
2. $\pi_i =$ ProvePSF$(sk_i.N, sk_i.\phi(N), Pk, P, q, i)$
3. EchoBroadcast π_i, pk_i to all other participants

Output: sk_i, pk_j

Algorithm 7 Gennaro/Goldfeder Paillier DKG Round 2

Adapted from the [GG20] Pseudocode

Input: $E, i, Pk, \{\pi_j, ..., \pi_n\}_{i \neq j}, \{pk_j, ..., pk_n\}_{i \neq j}, \{p_1, ..., p_n\}$

- E: elliptic curve as describe in the section 7
- i: positive integer for this participant.
- Pk: the public key computed in Algorithm 4.
- $\{\pi_j, ..., \pi_n\}_{i \neq j}$: the proofs generated in Algorithm 6 from other participants.
- $\{pk_j, ..., pk_n\}_{i \neq j}$: the paillier public keys generated in Algorithm 6 from other participants.
- $\{p_1, ..., p_n\}$: the set of participant ids as positive integers.

1. for j in $1, ..., n$
2. if VerifyPSF$(\pi_j, pk_j, Pk, P, q, p_j)$ = false ; $\perp$

Output: Broadcast Success

Algorithm 8 Gennaro/Goldfeder Proof Parameters DKG Round 1

Adapted from the [GG20] Pseudocode

Input: E, i

- E: elliptic curve as describe in the section 7
- i: positive integer for this participant.

1. Choose 1024-bit safe prime $P_i = 2p_i + 1$ where p_i is also prime.
2. Choose 1024-bit safe prime $Q_i = 2q_i + 1$ where q_i is also prime.
3. $N_i = P_i Q_i$
4. $f \xleftarrow{\$} Z_{N_i *}$
5. $\alpha \xleftarrow{\$} Z_{N_i *}$
6. $\beta = \alpha^{-1} p_i q_i$
7. $h_{1i} = f^2 \mod N_i$
8. $h_{2i} = h_{1i}^{\alpha} \mod N_i$
9. π_{1i} = ProveCompositeDL$(P, q, p_i, q_i, h_{1i}, h_{2i}, \alpha, N_i)$
10. π_{2i} = ProveCompositeDL$(P, q, p_i, q_i, h_{2i}, h_{1i}, \beta, N_i)$
11. EchoBroadcast $N_i, h_{1i}, h_{2i}, \pi_{1i}, \pi_{2i}$ to all other participants

Output: N_i, h_{1i}, h_{2i}

Algorithm 9 Gennaro/Goldfeder Proof Parameters DKG Round 2

Adapted from the [GG20] Pseudocode

Input: $E, i, \{\pi_{1j}, \pi_{2j}, ..., \pi_{nj}, \pi_{nj}\}_{i \neq j}, \{N_j, h_{1j}, h_{2j}\}_{i \neq j}$

- E: elliptic curve as describe in the section 7
- i: positive integer for this participant.
- $\{\pi_{1j}, \pi_{2j}, ..., \pi_{nj}, \pi_{nj}\}_{i \neq j}$ proofs sent by other participants generated in Algorithm 8
- $\{N_j, h_{1j}, h_{2j}\}_{i \neq j}$ the proof params send by other participants generated in Algorithm 8

1. for j in $1, ..., n$
2. if $i = j$ continue
3. if VerifyCompositeDL$(\pi_{1j}, P, q, h_{1j}, h_{2j}, N_j)$ = false ; $\perp$
4. if VerifyCompositeDL$(\pi_{1j}, P, q, h_{2j}, h_{1j}, N_j)$ = false ; $\perp$

Output: Broadcast Success

9 Composing algorithms for DKG

The flexibility these algorithms afford is that a secure and functional DKG can be implemented for each signature by only using what is necessary.

These same algorithms can be used to implement key resharing also known as (a.k.a.) proactive secret sharing (PSS). The only difference between DKG and PSS is how the secret value x is used. If x is generated randomly, this is a DKG for a new key. If x is provided from a previous DKG this is PSS.

9.1 BLS

BLS keys can be generated using Algorithms 2, 3, 4.

9.2 EDDSA or Schnorr

EDDSA or Schnorr keys can be generated using Algorithms 2, 3, 4.

9.3 ECDSA

ECDSA keys use various algorithms depending which protocol will be used. 2P-ECDSA uses Algorithms 2, 3, 4, 6, 7. [GG20] requires the same algorithms as 2P-ECDSA as well as 5, 8, 9.

References

[DKLas18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ecdsa from ecdsa assumptions. Cryptology ePrint Archive, Report 2018/499, 2018. https://eprint.iacr.org/2018/499.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, page 427–438, USA, 1987. IEEE Computer Society.

[GG20] Rosario Gennaro and Steven Goldfeder. One round threshold ecdsa with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. https://eprint.iacr.org/2020/540.

[GJKR05] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Journal of Cryptology*, pages 925–963, 05 2005.

[KG20] Chelsea Komlo and Ian Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. Internet-Draft draft-komlo-frost-00, Internet Engineering Task Force, August 2020. Work in Progress.

[Lin17] Yehuda Lindell. Fast secure two-party ecdsa signing. Cryptology ePrint Archive, Report 2017/552, 2017. https://eprint.iacr.org/2017/552.

[LNR18] Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. Cryptology ePrint Archive, Report 2018/987, 2018. https://eprint.iacr.org/2018/987.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.