Twisted ElGamal

Yu Chen

cycosmic@gmail.com

Abstract

In this work, we introduce a new additively homomorphic public-key encryption scheme called twisted ElGamal, which is not only as secure and efficient as standard exponential ElGamal, but also very friendly to zero-knowledge proof protocols, such as Sigma protocols and range proofs. This feature makes twisted ElGamal extremely useful in numerous privacy-preserving scenarios, such as cryptocurrencies with strong privacy and secure machine learning.

Contents

1 Introduction

In this report, we introduce twisted ElGamal [CMT19], which is additively homomorphic and friendly to zero-knowledge proofs.

2 Preliminaries

We first fix some notations and recall the related cryptographic assumptions.

2.1 Basic Notations

We denote the security parameter by $\lambda \in \mathbb{N}$. A function is negligible in λ, written $\mathsf{negl}(\lambda)$, if it vanishes faster than the inverse of any polynomial in λ. Let $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ be two distribution ensembles indexed by λ. We say that X and Y are statistically indistinguishable, written $X \approx_s Y$, if the statistical distance between X_λ and Y_λ is negligible in λ. We say that X and Y are computationally indistinguishable, written $X \approx_c Y$, if the advantage of any PPT algorithm in distinguishing X_λ and Y_λ is $\mathsf{negl}(\lambda)$. A probabilistic polynomial time (PPT) algorithm is a randomized algorithm that runs in time $\mathsf{poly}(\lambda)$. If $\mathcal{A}$ is a randomized algorithm, we write $z \leftarrow \mathcal{A}(x_1, \ldots, x_n; r)$ to denote that $\mathcal{A}$ outputs z on inputs $(x_1, \ldots, x_n)$ and randomness r. For notational clarity we usually omit r and write $z \leftarrow \mathcal{A}(x_1, \ldots, x_n)$. For a positive integer n, we use $[n]$ to denote the set of numbers $\{1, \ldots, n\}$. For a set X, we use $|X|$ to denote its size and use $x \xleftarrow{\text{R}} X$ to denote sampling x uniformly at random from X. We use U_X to denote the uniform distribution over X.

2.2 Cryptographic Assumptions

Let $\mathsf{GroupGen}$ be a PPT algorithm that on input a security parameter 1^λ, outputs description of a cyclic group $\mathbb{G}$ of prime order $p = \Theta(2^\lambda)$, and a random generator g for $\mathbb{G}$. In what follows, we describe the discrete-logarithm based assumptions related to $(\mathbb{G}, g, p) \leftarrow \mathsf{GroupGen}(1^\lambda)$.

Definition 2.1 (Decisional Diffie-Hellman Assumption)**.** The DDH assumption holds if for any PPT adversary, we have:

$$\left|\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is over the randomness of $\mathsf{GroupGen}(1^\lambda)$, $\mathcal{A}$'s random tape, and the random choices of $a, b, c \xleftarrow{\text{R}} \mathbb{Z}_p$.

Definition 2.2 (Divisible Decisional Diffie-Hellman Assumption)**.** The divisible DDH assumption holds if for any PPT adversary, we have:

$$\left|\Pr[\mathcal{A}(g, g^a, g^b, g^{a/b}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is over the randomness of $\mathsf{GroupGen}(1^\lambda)$, $\mathcal{A}$'s random tape, and the random choices of $a, b, c \xleftarrow{\text{R}} \mathbb{Z}_p$.

As proved in [BDZ03], the divisible DDH assumption is equivalent to the standard DDH assumption.

2.3 Public-Key Encryption

A PKE scheme consists of four polynomial time algorithms as follows.

- $\mathsf{Setup}(1^\lambda)$: on input a security parameter 1^λ, output public parameters pp. We assume that pp includes the descriptions of message space M, randomness space R.
- $\mathsf{KeyGen}(pp)$: on input pp, output a public key pk and a secret key sk.
- $\mathsf{Enc}(pk, m)$: on input a public key pk and a plaintext m, output a ciphertext C. When emphasizing the randomness r used for encryption, we denote this by $C \leftarrow \mathsf{Enc}(pk, m; r)$.

- $\mathsf{Dec}(sk, C)$: on input a secret key sk and a ciphertext C, output a plaintext m or a distinguished symbol $\perp$ indicating that C is invalid.

Remark 2.1. In practice, it is common for public parameters to be generated and fixed "once-and-for-all", and then shared by multiple users. Of course, each user can generate its own public parameters. In this case, Setup is absorbed into KeyGen and pp becomes a part of public key.

Correctness. For all $pp \leftarrow \mathsf{Setup}(1^\lambda)$, all $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$ and all $m \in M$ (here M is the message space), we have $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m$.

Homomorphism. For any public key pk, any $(m_1, r_1), (m_2, r_2) \in M \times R$, we have $\mathsf{Enc}(pk, m_1; r_1) + \mathsf{Enc}(pk, m_2; r_2) = \mathsf{Enc}(pk, m_1 + m_2; r_1 + r_2)$. Here, we slight abuse the symbol "+" to denote component-wise operation over ciphertext space $\mathbb{G}^2$. Additive homomorphism on message already suffices for most applications: $\mathsf{Enc}(pk, m_1) + \mathsf{Enc}(pk, m_2)$ is an encryption of $(m_1 + m_2)$ of pk under some appropriate randomness. When both ciphertext and message spaces are finite cyclic groups, we can naturally define scalar multiplication, which is a special case of additive homomorphism: for all $k \in \mathbb{Z}_p$, $k \cdot \mathsf{Enc}(pk, m)$ is an encryption of km of pk.

For clarity, we define extended algorithms of PKE as below:

- $\mathsf{ReRand}(sk, C; r)$: on input sk and c, output a re-randomized ciphertext C' of C.
- $\mathsf{HomoAdd}(C_1, C_2)$: on input C_1 and C_2, output $C = C_1 + C_2$.
- $\mathsf{HomoSub}(C_1, C_2)$: on input C_1 and C_2, output $C = C_1 - C_2$.
- $\mathsf{ScalarMul}(C, k)$: on input C and a scalar $k \in \mathbb{Z}_p$, output $C' = k \cdot C$.

IND-CPA security. Let $\mathcal{A}$ be an adversary against the semantic security of PKE component and define its advantage in the following experiment:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda); \\ (pk, sk) \leftarrow \mathsf{KeyGen}(pp) \\ (state, m_0, m_1) \leftarrow \mathcal{A}_1(pp, pk); \\ \beta \xleftarrow{\mathrm{R}} \{0, 1\}; \\ C^* \leftarrow \mathsf{Enc}(pk, m_\beta) \\ \beta' \leftarrow \mathcal{A}_2(state, C^*); \end{array}\right] - \frac{1}{2}.$$

We say a PKE scheme is IND-CPA secure if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the above security experiment.

Key Privacy/Anonymity. Let $\mathcal{A}$ be an adversary against the anonymity of the PKE component and define its advantage in the following experiment:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda); \\ (pk_0, sk_0), (pk_1, sk_1) \leftarrow \mathsf{KeyGen}(pp) \\ (state, m) \leftarrow \mathcal{A}_1(pp, pk_1, pk_2); \\ \beta \xleftarrow{\mathrm{R}} \{0, 1\}; \\ C^* \leftarrow \mathsf{Enc}(pk_\beta, m) \\ \beta' \leftarrow \mathcal{A}_2(state, C^*); \end{array}\right] - \frac{1}{2}.$$

We say a PKE scheme is anonymous (i.e., satisfying key privacy) if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the above security experiment.

3 Scheme Description and Security Analysis

3.1 Twisted ElGamal

Twisted ElGamal consists of four algorithms as below:

Basic algorithms:

- $\mathsf{Setup}(1^\lambda)$: run $(\mathbb{G}, g, p) \leftarrow \mathsf{GroupGen}(1^\lambda)$, pick $h \xleftarrow{R} \mathbb{G}^*$, set $pp = (\mathbb{G}, g, h, p)$ as global public parameters. The randomness and message spaces are $\mathbb{Z}_p$.
- $\mathsf{KeyGen}(pp)$: on input pp, choose $sk \xleftarrow{R} \mathbb{Z}_p$, set $pk = g^{sk}$.
- $\mathsf{Enc}(pk, m; r)$: compute $X = pk^r$, $Y = g^r h^m$, output $C = (X, Y)$.
- $\mathsf{Dec}(sk, C)$: parse $C = (X, Y)$, compute $h^m = Y/X^{sk^{-1}}$, recover m from h^m.

Extended algorithms:

- $\mathsf{ReRand}(sk, C; r')$: on input sk, $C = (X, Y)$ and randomness r', compute $h^m = Y/X^{sk^{-1}}$, then compute $X' = pk^{r'}$, $Y' = g^{r'} h^m$, output $C' = (X', Y')$.
- $\mathsf{HomoAdd}(C_1, C_2)$: on input $C_1 = (X_1, Y_1)$ and $C_2 = (X_2, Y_2)$, compute $X = X_1 + X_2$, $Y = Y_1 + Y_2$, output $C = (X, Y)$.
- $\mathsf{HomoSub}(C_1, C_2)$: on input $C_1 = (X_1, Y_1)$ and $C_2 = (X_2, Y_2)$, compute $X = X_1 - X_2$, $Y = Y_1 - Y_2$, output $C = (X, Y)$.
- $\mathsf{HomoScalar}(C, k)$: on input C and a scalar $k \in \mathbb{Z}_p$, compute $X' = k \cdot X$, $Y' = k \cdot Y$, output $C' = (X', Y')$.

Correctness and additive homomorphism are obvious. The IND-CPA security and anonymity are established by the following two theorems.

Theorem 3.1. *Twisted ElGamal is IND-CPA secure based on the divisible DDH assumption.*

Proof. We proceed via two games. Let S_i be the probability that $\mathcal{A}$ wins in Game i.

Game 0. The real IND-CPA security experiment. Challenger $\mathcal{CH}$ interacts with $\mathcal{A}$ as below:

1. Setup: $\mathcal{CH}$ generates pp and keypair as per definition, then sends pp and $pk = g^{sk}$ to $\mathcal{A}$.
2. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{CH}$ as the target messages. $\mathcal{CH}$ picks a random bit β and randomness r, computes $X = pk^r$, $Y = g^r h^{m_\beta}$, sends $C = (X, Y)$ to $\mathcal{A}$.
3. Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

According to the definition of Game 0, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_0] - 1/2.$$

Game 1. Same as Game 0 except $\mathcal{CH}$ generates the challenge ciphertext in a different way.

2. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{CH}$ as the target messages. $\mathcal{CH}$ picks a random bit β and two independent randomness r and s, computes $X = pk^r$, $Y = g^s h^{m_\beta}$, sends $C = (X, Y)$ to $\mathcal{A}$.

In Game 1, the distribution of C is independent of β, thus we have:

$$\Pr[S_1] = 1/2.$$

It remains to prove $\Pr[S_1]$ and $\Pr[S_0]$ are negligibly close. We prove this by showing if not so, one can build an adversary $\mathcal{B}$ breaks the divisible DDH assumption with the same advantage. Given (g, g^a, g^b, g^c), $\mathcal{B}$ is asked to decide if it is a divisible DDH tuple or a random tuple. To do so, $\mathcal{B}$ interacts with $\mathcal{A}$ by simulating $\mathcal{A}$'s challenger in the following IND-CPA experiment.

1. Setup: $\mathcal{B}$ picks $h \xleftarrow{R} \mathbb{G}^*$, sets $pp = (\mathbb{G}, g, h, p)$, sets g^b as the public key, where $b \in \mathbb{Z}_p$ is interpreted as the corresponding secret key, which is unknown to $\mathcal{B}$. $\mathcal{B}$ sends pp and $pk = g^b$ to $\mathcal{A}$.
2. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{B}$. $\mathcal{B}$ picks a random bit β and sets $X = g^a$, $Y = g^c h^{m_\beta}$, sends $C = (X, Y)$ to $\mathcal{A}$.
3. Guess: $\mathcal{A}$ outputs a guess β'. $\mathcal{B}$ outputs "1" if $\beta' = \beta$ and "0" otherwise.

If (g, g^a, g^b, g^c) is a divisible DDH tuple, $\mathcal{B}$ simulates Game 0 perfectly (with randomness $c = a/b$). Else, $\mathcal{B}$ simulates Game 1 perfectly (with two independent randomness a/b and c). Thereby, we have $\mathsf{Adv}_{\mathcal{B}} = |\Pr[S_0] - \Pr[S_1]|$, which is negligible in λ by the divisible DDH assumption.

Putting all the above together, Theorem 3.1 follows. □

Remark 3.1. We stress that twisted ElGamal remains secure even the adversary knows the discrete relation between (g, h). Thereby, h can be set as a fixed or random element in $\mathbb{G}^*$. In both cases, twisted ElGamal does not require trusted setup. Looking ahead, we set h as a hash value of g in our implementation, which is in line with the transparent setup in Bulletproofs [BBB+18].

	size				efficiency		
	pp	pk	sk	C	KeyGen	Enc	Dec
standard ElGamal	$\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$\|\mathbb{Z}_p\|$	$\|2\mathbb{G}\|$	1Exp	3Exp+2Add	1Exp+1Add+1DLOG
twisted ElGamal	$2\|\mathbb{G}\|$	$\|\mathbb{G}\|$	$\|\mathbb{Z}_p\|$	$\|2\mathbb{G}\|$	1Exp	3Exp+2Add	1Exp+1Add+1DLOG

Table 1: Comparison between standard ElGamal and twisted ElGamal. $|\mathbb{Z}_p|$ denotes the bits of a scalar field element, $|\mathbb{G}|$ denotes the bits of a group element. Exp and Add denote exponentiation and add operation in $\mathbb{G}$, while DLOG denotes discrete search operation in $\mathbb{G}$.

Theorem 3.2. *Twisted ElGamal is anonymous based on the divisible DDH assumption.*

Proof. We proceed via four games.

Game 0. Challenger $\mathcal{CH}$ interacts with $\mathcal{A}$ as follows:

1. Setup: $\mathcal{CH}$ generates pp and two keypairs as per definition, then sends pp and pk_1 and pk_2 to $\mathcal{A}$.
2. Challenge: $\mathcal{A}$ submits m to $\mathcal{CH}$ as the target message. $\mathcal{CH}$ picks a randomness r, computes $X = pk_1^r$, $Y = g^r h^m$, sends $C = (X, Y)$ to $\mathcal{A}$.
3. Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

Game 1. Same as Game 0 except $\mathcal{CH}$ generates the challenge ciphertext in a different way.

2. Challenge: $\mathcal{A}$ submits m to $\mathcal{CH}$ as the target message. $\mathcal{CH}$ picks a random bit β and two independent randomness r and s, computes $X = pk_1^r$, $Y = g^s h^m$, sends $C = (X, Y)$ to $\mathcal{A}$.

Game 2. Same as Game 1 except $\mathcal{CH}$ generates ciphertext using pk_2.

2. Challenge: $\mathcal{A}$ submits m to $\mathcal{CH}$ as the target message. $\mathcal{CH}$ picks a random bit β and two independent randomness r and s, computes $X = pk_2^r$, $Y = g^s h^m$, sends $C = (X, Y)$ to $\mathcal{A}$.

Game 3. Same as Game 2 except $\mathcal{CH}$ generates ciphertext under the same randomness.

2. Challenge: $\mathcal{A}$ submits m to $\mathcal{CH}$ as the target message. $\mathcal{CH}$ picks randomness r, computes $X = pk_2^r$, $Y = g^r h^m$, sends $C = (X, Y)$ to $\mathcal{A}$.

It is easy to see that Game 0 $\approx_c$ Game 1, Game 2 $\approx_c$ Game 3 are computationally indistinguishable based on the divisible DDH assumption, while Game 1 $\equiv$ Game 2. Putting all the above together, Theorem 3.2 follows. □

Recall that the divisible DDH assumption is equivalent to the standard DDH assumption, thus we conclude that twisted ElGamal is IND-CPA secure and anonymous based on the standard DDH assumption. Moreover, both of the security reductions are tight.

4 Optimizations and Discussions

In this section, we describe optimizations and summarize the advantages of twisted ElGamal.

4.1 Faster Decryption Algorithm

As in standard exponential ElGamal, to ensure additive homomorphism on message space $\mathbb{Z}_p$, message in twisted ElGamal is also encoded to the exponent. This makes decryption inefficient even with knowledge of secret key. The bottleneck is the interval-version discrete logarithm search algorithm. Let $[0, 2^\ell)$ be the interval (message space). The time complexity of brute-force is $O(2^\ell)$. We recommend employing dedicated algorithms to compute a discrete logarithm in an interval more efficiently. The candidate algorithms include:

- Pollard's kangaroo method [Pol78] and Galbraith-Ruprai algorithm [GR10]: run in time $O(2^{\ell/2})$ with constant memory cost.
- Bernstein-Lange algorithm [BL12]: runs in time $O(2^{\ell/3})$ using a table of size $O(2^{\ell/3})$.
- Shanks's algorithm [Sha71]: runs in time $O(2^{\ell/2})$ using a table of size $O(2^{\ell/2})$.

In this submission, we choose Shanks's algorithm for the following reasons: (i) the first two algorithms run in expected polynomial time, namely, they may take long time to find the solution in worst cases; (ii) Shanks's algorithm is simple and admits flexible time-space trade off. Let n be an integer between $[0, \ell/2]$, it runs in time $O(2^{\ell/2-n})$ using a table of size $O(2^{\ell/2+n})$. We refer to n as the time-space tuning parameter, which is set to be 0 in the standard Shanks's algorithm.

4.2 Signature and Zero-Knowledge Friendly

In many privacy-preserving applications, homomorphic PKE is used in combination with other cryptographic schemes. As we will show, twisted ElGamal enjoys nice interoperability with signature and zero-knowledge proofs.

Integrate with signature. The keypair of twisted ElGamal is a discrete logarithm tuple, which is identical to widely used signature in practice, such as ECDSA and Schnorr signature. This fact enables efficient design of integrated signature and encryption schemes by composing twisted ElGamal and ECDSA/Schnorr. Please refer to [CMT19] for more technical details, include formal security proof and example application.

Work with zero-knowledge proofs. The algebra structure of twisted ElGamal admits simple and efficient Sigma protocols for proving knowledge of plaintext and randomness, two ciphertexts encrypt the same message. Moreover, we can design efficient accompanying range proofs by directly invoking the state-of-the-art Bulletproofs [BBB+18]. This is superior than the standard ElGamal PKE. To see why, recall that Bulletproofs only accepts statements of the form of Pedersen commitment $g^r h^m$. To guarantee soundness, the DL relation between commitment key (g, h) must be unknown to the prover. However, a standard ElGamal ciphertext C of m under pk is of the form $(g^r, pk^r g^m)$. Although the second component can be viewed as a commitment of m under commitment key (pk, g), we cannot *directly* employ Bulletproofs to generate range proof for m, because sk constitutes an obvious trapdoor. In contrast, the second component of twisted ElGamal ciphertext is exactly of the form of Pedersen commitment $g^r h^m$, and the DL relation between (g, h) is unknown. This allows us to directly invoke Bulletproofs to generate range proofs for encrypted values under twisted ElGamal in a black-box manner, and these proofs can be easily aggregated. Please refer to [CMT19] for more technical details, including useful gadgets and formal security proof.

We note that we do can employ Bulletproofs to generate range proof for messages encrypted by standard ElGamal PKE. But, we need to use a Pedersen commitment as a bridge, i.e., (1) generate a Pedersen commitment $g^r h^m$; (2) prove the randomness and message under ElGamal ciphertext $(g^r, pk^r h^m)$ and Pedersen commitment $g^r h^m$ are consistent via a Sigma protocol; (3) invoke Bulletproofs on $g^r h^m$. By using twisted ElGamal in the place of standard ElGamal, we are able to dismiss steps (1) and (2). We compare the cost of working with Bulletproofs between standard and twisted ElGamal PKE in the following table.

	size	efficiency
standard ElGamal	$2\|\mathbb{G}\| + \|\mathbb{Z}_p\|$	4Exp+1Add
twisted ElGamal	0	0

Table 2: The costs of working with Bulletproofs between standard ElGamal and twisted ElGamal. The cost of ElGamal working with Bulletproofs comes from two parts, i.e., an additional Pedersen commitment and a Sigma protocol for consistency between Pedersen commitment and ElGamal ciphertext.

5 Implementation and Evaluation

We report an implementation of twisted ElGamal in C++ based on OpenSSL, and collect the benchmarks on a MacBook Pro with an Intel i7-4870HQ CPU (2.5GHz) and 16GB of RAM, using elliptic curve NID_X9_62_prime256v1[1] [Ope], which has 128 bit security. The default message space is 40bit, i.e., $[0, 2^{40} - 1)$. Enc and Scalar in 2 thread setting are roughly $15\% \sim 25\%$ faster than their counterpart in the single thread setting.

	hash map	Setup	KeyGen	Enc	Dec	ReRand	Add	Sub	Scalar
1 thread	275MB	50s+6s	0.0158	0.141	400	0.157	0.0031	0.0039	0.136
	550MB	100s+12s	—	—	200	—	—	—	—
2 thread	—	—	0.116	—	331	—	—	—	0.094
4 thread	275MB	—	—	—	173	—	—	—	—
4 thread	1.1GB	—	—	—	42	—	—	—	—

Table 3: Except the Setup algorithm (highlighted in red color), the efficiency of other algorithms are measured by ms. The decryption time is obtained using a hash map of size roughly 270MB. Faster decryption speed can be achieved by enlarging the table. The setup algorithm runs "one-and-for-all", it first generates and serializes a key-value hash map called "`point2index.map`" to accelerate decryption (this step takes roughly 50s), then loads it to the memory (this step takes roughly 6s).

References

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018*, pages 315–334. IEEE Computer Society, 2018.

[BDZ03] Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of diffie-hellman problem. In *Information and Communications Security, 5th International Conference, ICICS 2003*, volume 2836 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2003.

[BL12] Daniel J. Bernstein and Tanja Lange. Computing small discrete logarithms faster. In *Progress in Cryptology - INDOCRYPT 2012*, volume 7668 of *Lecture Notes in Computer Science*, pages 317–338. Springer, 2012.

[CMT19] Yu Chen, Xuecheng Ma, and Cong Tang. PGC: Pretty Good Confidential Transaction System with Accountability. Cryptology ePrint Archive, Report 2019/319, 2019. https://eprint.iacr.org/2019/319.

[GR10] Steven D. Galbraith and Raminder S. Ruprai. Using equivalence classes to accelerate solving the discrete logarithm problem in a short interval. In *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 368–383. Springer, 2010.

[Ope] https://github.com/openssl/openssl/blob/4e6647506331fc3b3ef5b23e5dbe188279ddd575/include/openssl/obj_mac.h.

[Pol78] John M. Pollard. Monte carlo methods for index computation (mod p). *Mathematics of Computation*, 32, 1978.

[1] Our implementation also supports other elliptic curves.

[Sha71] D. Shanks. Class number, a theory of factorization and genera. *Proc. Symposium in Pure Mathematics*, 20, 1971.