Anamorphic Cryptography with Elliptic Curve Methods

William J. Buchanan[1]

Blockpass ID Lab, Edinburgh Napier University, Edinburgh.

Abstract. In 2022, Persianom, Phan and Yung outlined the creation of Anamorphic Cryptography. With this, we can create a public key to encrypt data, and then have two secret keys. These secret keys are used to decrypt the cipher into different messages. So, one secret key is given to the Dictator (who must be able to decrypt all the messages), and the other is given to Alice. Alice can then decrypt the ciphertext to a secret message that the Dictator cannot see. This paper outlines the implementation of Anamorphic Cryptography using ECC (Elliptic Curve Cryptography), such as with the secp256k1 curve. This gives considerable performance improvements over discrete logarithm-based methods. Overall, it outlines how the secret message sent to Alice is hidden within the random nonce value, which is used within the encryption process, and which is cancelled out when the Dictator decrypts the ciphertext.

1 Introduction

In cybersecurity, we can use anamorphic cryptography to change the viewpoint of a cipher [1]. With this, we assume that we have a dictator who will read all of our encrypted data, and will thus have a secret key of sk. The dictator (Mallory) will arrest anyone who sends secret messages that they cannot read. For this, Bob could create two secret keys: sk_0 and sk_1. He sends sk_0 to Mallory and sk_1 to Alice (the person to whom Bob wants to send a secret message). As far as Mallory knows, he has the only key for the ciphertext. Dodis et al [2] liken this this approach to the adding of backdoors into semantically secure schemes is pointless, and where entities might be asked to hand-over their decryption keys.

2 Background

With anamorphic encryption, we can have a public key of pk and two private keys of $sk0$ and $sk1$. Bob then can have two messages of:

$$m_0 = \text{"I love the Dictator"} \tag{1}$$
$$m_1 = \text{"I hate the Dictator"} \tag{2}$$

Bob then encrypts the two messages with the public key:

$$CT = Enc(pk, m_0, msg_1) \tag{3}$$

The Dictator will then decrypt with sk_0 and reveal the first message:

$$Dec(sk_0, CT) \rightarrow m_0 \tag{4}$$

Alice will decrypt with her key and reveal the second message:

$$Dec(sk_1, CT) \rightarrow m_1 \tag{5}$$

And, so, the Dictator thinks that they can decrypt the message, and gets, "I love the Dictator". Alice, though, is able to decrypt the ciphertext to a different message of "I hate the Dictator".

Carnemolla et al. [3] have outlined that there are certain schemes which are *Anamorphic Resistant* and that, in some cases, anamorphic encryption is similar to substitution attacks. The methods that support anamorphic encryption include RSA-OAEP, Pailler, Goldwasser-Micali, ElGamal schemes, Cramer-Shoup, and Smooth Projective Hash-based systems.

In this case, we will use the ElGamal method [4]. An implementation of the ElGamal method for anamorphic cryptography is given in [5]. While discrete logarithm methods have been used to implement Anamorphic Cryptography [6], they tend to be slow in their operation. This paper outlines the integration of ElGamal methods with ECC for the implementation of Anamorphic Encryption.

3   ElGamal encryption

With ElGamal encryption using elliptic curves [7], Alice generates a private key (x) and a public key of:

$$Y = x.G \tag{6}$$

and where G is the base point on the curve. She can share this public key (Y) with Bob. When Bob wants to encrypt something for Alice, he generates a random value (r) and the message value (M) and then computes:

$$C_1 = r.G \tag{7}$$
$$C_2 = r.Y + M \tag{8}$$

To decrypt, Alice takes her private key (x) and computes:

$$M = C_2 - x.C_1 \tag{9}$$

This works because:

$$M = C_2 - y.C_1 = r.x.G + M - x.r.G = M \tag{10}$$

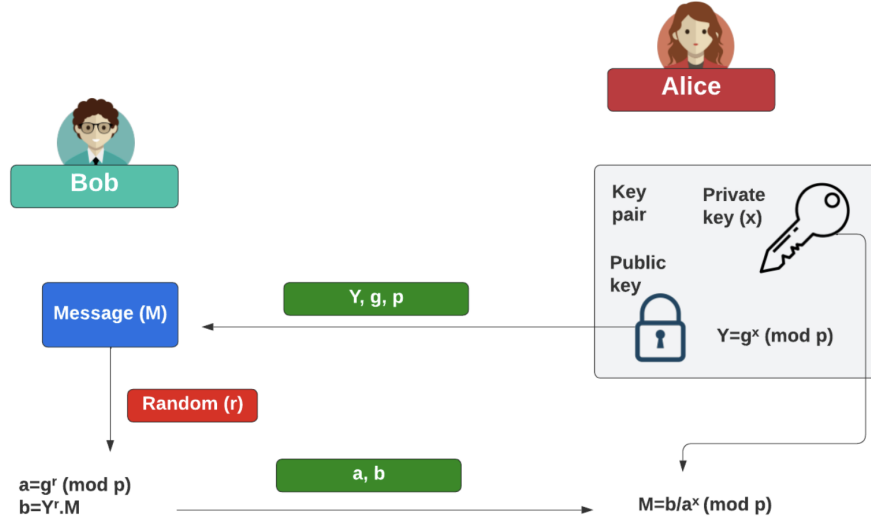Figure 1 outlines how Bob can encrypt data for Alice.

Fig. 1. ElGamal encryption with ECC

4 Anamorphic Encryption with ECC

First, we pick a curve, such as secp256k1, which has a base point of G. Bob can then pick a secret key for the Dictator of $sk_{Dictator}$. The public key is then:

$$pk = sk_{Dictator}.G \tag{11}$$

Bob then generates a random scalar value of t and takes the secret message of cm, and produces:

$$r = cm + t \tag{12}$$

The value of t will be Alice's secret key. To encrypt the message of m, Bob uses:

$$rY = r.pk \tag{13}$$
$$rG = r.G \tag{14}$$
$$rYval = \text{Int}(rY) \tag{15}$$
$$c0 := rYval + M \tag{16}$$
$$c1 := rG \tag{17}$$
$$\tag{18}$$

The cipher is the (c_0, c_1). To decrypt by the Dictator:

$$yC = sk_{Dictator}.c1 \tag{19}$$
$$yC_{val} = \text{Int}(yC) \tag{20}$$
$$res_{Dictator} = c_0 - yC_{val} \tag{21}$$
$$\tag{22}$$

Alice can then decrypt with her key (t):

$$tc = t.G \tag{23}$$
$$res_{Alice} = c_1 - tc \tag{24}$$

Alice will then search through the possible values of res_{Alice} to find the value of cm that matches the elliptic curve point. This works because:

$$res_{Alice} = c_1 - tc = r.G - t.G = r.G - (r - cm).G = cm.G \tag{25}$$

Some sample code and a test run are given in the appendix.

5   Appendix

Some sample code is [8]:

```
package main

import (
        crand "crypto/rand"
        "fmt"
        "math/big"
        "math/rand"
        "os"
        "strconv"
        "time"

        "github.com/coinbase/kryptology/pkg/core/curves"
)



func main() {
        rand.Seed(time.Now().UnixNano())

        argCount := len(os.Args[1:])

        x := 5
        cm:=99
```

```go
        if argCount > 0 {
                x, _ = strconv.Atoi(os.Args[1])
        }

        if argCount > 1 {
                cm, _ = strconv.Atoi(os.Args[2])
        }

        rand.Seed(time.Now().UnixNano())

        curve := curves.K256()
        G := curve.Point.Generator()

        sk_Dictator := curve.Scalar.Random(crand.Reader)

        pk := G.Mul(sk_Dictator)

        t:=curve.Scalar.Random(crand.Reader)

        r := curve.Scalar.New(cm).Add(t)

        fmt.Printf("\n\nDictator key: %d\n",sk_Dictator.BigInt())

        fmt.Printf("Alice key: %d\n\n",t.BigInt())


// Encrypt
        rY := pk.Mul(r)
        rG := G.Mul(r)
        rYval := new(big.Int).SetBytes(rY.ToAffineUncompressed())
        c0 := new(big.Int).Add(rYval, big.NewInt(int64(x)))
        c1 := rG

        fmt.Printf("Encrypted (c0): %s\n",c0)
        fmt.Printf("Encrypted (c1): %x\n\n",c1.ToAffineUncompressed())

// Decrypt by Dictator

        yC := c1.Mul(sk_Dictator)
        yCval := new(big.Int).SetBytes(yC.ToAffineUncompressed())
        res_Dictator := new(big.Int).Sub(c0, yCval)

        fmt.Printf("Dictator message: %d\n",x)
        fmt.Printf("Dictator recovered: %d\n",res_Dictator)

// Decrypt by Alice

        tc1 := G.Mul(t)
        res_Alice := c1.Sub(tc1)
```

```
        for i:=0;i<=1000;i++ {
                res:=G.Mul(curve.Scalar.New(i))

                if (res_Alice.Equal(res)==true) {
                        fmt.Printf("Alice␣message:␣%d\n",cm)
                        fmt.Printf("Alice␣recovered:␣%d\n",i)
                        break
                }

        }
}
```

A sample run which a message of 6 for the Dictator and a message of 20 for Alice, is:

```
Dictator key: 4943740139441239174205314912092041934903506
23590197131460183087403983256305661
Alice key: 377340532826187077325935517076113677691202962871734897107
90302858004991636924

Encrypted (c0): 55563164817718419919868920306927066318894666197272261307
89262842153481756082556801797346653416320499802923457710
4809798477077805759840884748908002510518326
Encrypted (c1): 0456281d59ee248ad030b49d86b1a9d45652c72669b05e914d22931b
fbdca6bc0601fedf56ced690a5cf0aa15901458e427f1b9055b16175
60ce210495b9e78cf1

Dictator message: 6
Dictator recovered: 6
Alice message: 20
Alice recovered: 20
```

6   Conclusions

While RSA-OAEP, Pailler, Goldwasser-Micali, ElGamal schemes, Cramer-Shoup, and Smooth Projective Hash-based systems all support anamorphic cryptography, the usage of elliptic curve methods provides an opportunity to enhances the overall performance of the methods implemented for the ElGamal technique.

References

1. G. Persiano, D. H. Phan, and M. Yung, "Anamorphic encryption: private communication against a dictator," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2022, pp. 34–63.

2. Y. Dodis and E. Goldin, "Anamorphic-resistant encryption; or why the encryption debate is still alive," *Cryptology ePrint Archive*, 2025.

3. D. Carnemolla, D. Catalano, E. Giunta, and F. Migliaro, "Anamorphic resistant encryption: the good, the bad and the ugly," *Cryptology ePrint Archive*, 2025.

4. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

5. F. Banfi, K. Gegier, M. Hirt, U. Maurer, and G. Rito, "Anamorphic encryption, revisited," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2024, pp. 3–32.

6. fbanfi90, "robust-anamorphic-encryption/elgamal.py at main · fbanfi90/robust-anamorphic-encryption — github.com," https://github.com/fbanfi90/robust-anamorphic-encryption/blob/main/elgamal.py, [Accessed 21-04-2025].

7. W. J. Buchanan, "Elgamal ecc encryption (using message string)," https://asecuritysite.com/elgamal/elgamal02_str, Asecuritysite.com, 2025, accessed: April 21, 2025. [Online]. Available: https://asecuritysite.com/elgamal/elgamal02_str

8. ——, "Anamorphic cryptography with elliptic curve cryptography," https://asecuritysite.com/principles_pub/ana2, Asecuritysite.com, 2025, accessed: April 21, 2025. [Online]. Available: https://asecuritysite.com/principles_pub/ana2