International Journal of Computer Science and Software Engineering (IJCSSE), Volume 5, Issue 11, November 2016
ISSN (Online): 2409-4285 www.IJCSSE.org Page: 248-252

# Short and Secure CLS Pattern Using Simple Crypto Analysis Hash Technique

**Ashok Kumar[1] and Kalyani Dasari[2]**

[1] M-Tech (CNIS), [2]Assistant Professor, Department of IT, VNRVJIET, Hyderabad

**ABSTRACT**

In the irregular oracle miniature under the hardness presumptions of K-CAA and Inv-CDHP we introduce a CLS (Certificateless Signature) which proved to be a much secured in traditional public key cryptosystem (PKC). We overcome the incompetent MaptoPoint hash technique by replacing a simple cryptoanalysis hash technique by indulging most common properties of CLS conspires. The extent of signatures generated in this paper is nearly 160 bits, which strength our assumption towards less calculation cost and essentially more productive than every single known CLS plans. In this way it can be utilized generally and particularly in low-data transmission correspondence situations.

Keywords: *Crypto Analysis, CLS Pattern, Hash Technique.*

## 1. INTRODUCTION

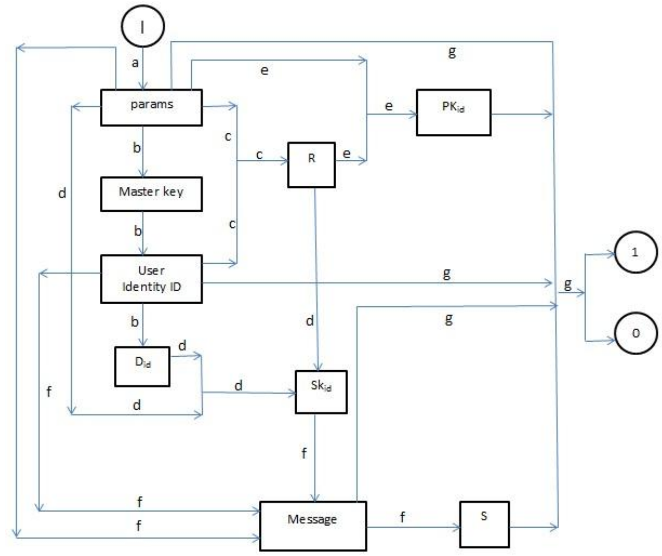
To maintain a strategic distance from the innate key escrow issue in ID-based open key cryptosystem, Al-Riyami and Paterson [2] presented another approach called certificateless open key cryptography (CLPKC) in 2003. The CLPKC is transitional between conventional PKC and ID-based cryptosystem. In a certificateless cryptosystem, a client's private key is not created by the PKG alone. Rather, it comprises of fractional private key created by the Key Generation Center (KGC) and some mystery esteem picked by the client. Along these lines, the KGC can't get the client's private key. In a manner that the key escrow issue can be tackled. Intuitionally, CLPKC has pleasant elements obtained from both ID-based cryptography and conventional PKC. It lightens the key escrow issue in ID-based cryptography and in the meantime decreases the cost and disentangles the utilization of the innovation when contrasted and conventional PKC. In a conventional open key cryptosystem (PKC), any individual who needs to send messages to others must get their approved declarations that contain people in general key. Nonetheless, this necessity brings loads of declaration administration issues by and by. With a specific end goal to maintain a strategic distance from the issues and the cost of appropriating people in general keys, Shamir [1] firstly presented the idea of personality based open key cryptosystem in 1984, which permits a client to utilize his personality data, for example, name, Email address, IP address or phone number, and so on as his own open key. It implies that there is no requirement for a client to keep an open key catalog or acquire other clients' authentications before correspondence. Be that as it may, there exists an inalienable disadvantage called private key escrow issue in an ID-based open key cryptosystem. Since this cryptosystem includes a Private Key Generator (PKG), which is in charge of producing a client's private key in light of his personality. Thus, the PKG can actually unscramble any cipher text or produce any client's mark on any message.

Generally, the PKI suffers two problems, namely: scalability and certificate management. The Identity-based Public Key Cryptography (IDPKC) came to address these two problems, but could not offer true nonrepudiation due to the key escrow problem. In ID-PKC, an entity's public key is derived directly from certain aspects of its identity, for example, an IP address belonging to a network host, or an e-mail address associated with a user. Private keys are generated for entities by a trusted third party called a private key generator (PKG). The first fully practical and secure identity-based public key encryption scheme was presented. Since then, rapid development of ID-PKC has taken place. Currently, there exist Identity-based Key Exchange protocols (interactive as well as noninteractive), signature schemes, and Hierarchical schemes. It has also been illustrated how ID-PKC can be used as a tool to enforce what might be termed "cryptographic work-flows", that is, sequences of operations (e.g. authentications) that need to be performed by an entity in order to achieve a certain goal. In 2003 Al-Riyami and Paterson introduced the concept of Certificateless Public Key Cryptography (CL-PKC) to overcome the key escrow limitation of the identity-based public key cryptography (ID-PKC). In CL-PKC a trusted third party called Key Generation Center (KGC) supplies a user with a partial private key. Then, the user combines the partial private key with a secret value (that is unknown to the KGC) to obtain his full private key. In this way the KGC does not know the user's private key. Then the user combines his secret value with the KGC's public parameters to compute his public key. The CL-PKC is considered a combination between PKI and identity based cryptography. It combines the best features of the PKI and IDPKC, such as no key escrow property, reasonable trust to trust authority and lightweight infrastructure. It provides a solution to the non-repudiation problem, through enabling a user to generate his/her full long-term private key, where the trusted third party is unable to impersonate the user. The use of certificateless cryptography schemes have appeared in literature, this includes the uses of certificateless encryption Certificateless signatures and certificateless signcryption. Al-Riyami and Paterson scheme proposed binding technique to link the public key by one-to-one correspondence with the identity to guarantee that every user in the system has one public/private key pair, the big contribution of using this binding technique is that upgrade the CL-PKC to trust level 3 as Girault's definition of the trust levels. Al-Riyami and Paterson proved that their certificateless encryption scheme is secure against fully-adaptive chosen cipher text attack (IND-CCA) and also proposed certificateless digital signature scheme along with certificateless key agreement protocol and hierarchal certificateless encryption scheme (HCLPKE).

In the CLS plans, an uncommon hash work called MapToPoint work which is utilized to outline character data into a point on elliptic bend is required. In any case, the hash capacity is wasteful in spite of the fact that there has been much examination on the development of such hash calculation. Consequently, utilizing general cryptographic hash work rather than the MapToPoint capacity can make strides the productivity of CLS plans. At present, numerous short marks conspires in conventional PKC have been proposed since Boneh et al. develop a short signature called BLS signature, which is simply a large portion of the measure of the mark in DSA (320-bits) with equivalent security. As a result of the little size of short marks, they are required in situations with stringent transmission capacity imperatives, for example, bar-coded computerized marks on postage stamps. By the by, to our best information, no short CLS plans have been discovered in this way. Certificateless marks produced by plans have roughly 320-bits sizes and marks in have no less than 480-piece sizes if utilizing an elliptic bend on $F_{397}$.Henceforth, it's fundamental for us to develop a short CLS plot.

# 2. FRAMEWORK & IMPLEMENTATION

Proposed framework has below mentioned algorithms which formulate the CLS scheme.




***System Setup:***
Building a system of frames like param's and master key by considering as security parameter l.

***Initial-Private- Key-retriever:***
Retrieving initial private key by taking inputs as param's, master key and user's identity

***Secret-pass-key-generator:***
To generate a secret pass key (r) by considering param's and a user's identity ID

***Private-Key-generator:***
To generate a private key ($SK_{id}$) by taking the values initial private key , private key .

***Public-Key-generator:***
To generate a public key by considering param's and secret pass key.

***Signature-generator:***
To generate Signature (S) by using param's, message, user's identity and private key

***Signature-verifier:***
By using param's, a public key $PK_{id}$, a message m, a user's identity ID, and a signature S, as input values and returns 1 means that the signature is accepted. Otherwise, 0 means rejected.

Pairing scheme which is used for this analysis is bilinear pairing technique, which is mentioned as below.

Bilinear pairing is a map e: $G1\times G1\rightarrow G2$ which satisfies the following properties, by considering G1 as a cyclic additive group of prime order q, and G2 as a cyclic multiplicative group of the same order q.

***(1) Bilinearity***
e(aP, bQ)=e(P, Q)ab, where P, QaG1, a, baZq*.
***(2) Non-degeneracy***
There exists P, QaG1 such that e(P, Q)≠1.
***(3) Computability***
There is an efficient algorithm to compute e(P, Q) for all P, Q a G1. Bilinear pairing happens by considering modified Tate or Weil pairing on super singular elliptic curve.

## 3. PROPOSED SYSTEM

Below is the theoretical sample code which uses param’s, a public key PKid, a message m, a user's identity ID, and a signature S, as input values and returns 1 means that the signature is accepted. Otherwise, 0 means rejected.

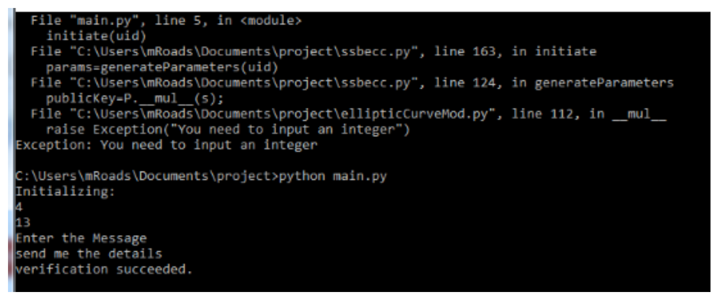
```
#define polynomials under the form of:
#a + b*x + c*x^2 + ...
class Polynomial(object):
    def __init__(self, c, p):
        if type(c) is Polynomial:
        self.coefficients=c.coefficients
        elif isinstance(c, ModP):
            self.coefficients = [c]
        elif not hasattr(c, '__iter__') and not
hasattr(c, 'iter'):
            self.coefficients=[ModP(c,p)]
        else:
            self.coefficients = c
        self.p = p
        self.coefficients=
strip(self.coefficients, ModP(0,p))
        self.name = '(Z/%dZ)[x]' % p
    #check if the polynomial is 0
    def isZero(self):
        return self.coefficients == []
    #function to print the polynomial
    def __repr__(self):
        if self.isZero():
            return '0'
        #iterate through the list of coefficients
and add them to one string
        else:
            return ' + '.join(['%s x^%d' %
(a,i) if i>0 else '%s' % a for i,a in
enumerate(self.coefficients)])
    #length of the polynomial
    def __abs__(self):
        return len(self.coefficients)
    #length of the polynomial
    def __len__(self):
        return len(self.coefficients)
```

```
    #subtract to polynomials by subtracting their
coeff.
    def __sub__(self, other):
        return self + (-other)
    def __rsub__(self, other):
        return -self + other
    #iterate through the coefficients
    def __iter__(self):
        return iter(self.coefficients)
    #negative of a polynomial
    def __neg__(self):
        return Polynomial([-a for a in
self],self.p)
    #iterate through polynomial
    def iter(self):
        return self.__iter__()
    #the leading coefficient of a polynomial
    def leadingCoefficient(self):
        return self.coefficients[-1]
    #the degree of a polynomial, ie largest
exponent
    def degree(self):
        return abs(self)-1
    #check whether two polynomials are equal or
not by comparing coefficients and same degree
    def __eq__(self,other):
        return self.degree() == other.degree()
and all([x==y for (x,y) in zip (self,other)])
    #add two polynomials by adding their
coefficients
    def __add__(self,other):
        #if integer, than one needs to make a
constant polynomial
        if isinstance(other, int):

    other=Polynomial([other],self.p)
        #adding the coefficients together.
fillvalue defines the value to use if one polynomial
  #has a smaller degree than the other one.
        newCoefficients = [sum(x) for x in
itertools.zip_longest(self,other, fillvalue=
ModP(0,self.p))]
        return Polynomial(newCoefficients,
self.p)
    def __radd__(self, other):
        return self + other
    #multiplication of two polynomials
    def __mul__(self,other):
        if isinstance(other, int):
            return
self*Polynomial([other],self.p)
        if self.isZero() or other.isZero():
            return Zero(self.p)
        else:
            #set all coefficients to zero
```

```
        newCoefficients= [ModP(0,self.p) for _ in range(len(self)+ len(other) - 1)]
        #general formula for the coefficients of the multiplication of two poly.
        for i,a in enumerate(self):
            for j,b in enumerate(other):

                newCoefficients[i+j] = newCoefficients[i+j] + a*b
        return Polynomial(newCoefficients,self.p)
    def __rmul__(self, other):
        return self * other
    #divmod for polynomials
    def __divmod__(self,divisor):
        quotient = Zero(self.p)
        remainder = self
        divisorDeg = divisor.degree()
    divisorLC=divisor.leadingCoefficient()
        while remainder.degree() >= divisorDeg:
        StockExponent=remainder.degree() - divisorDeg
            StockZero = [ModP(0,self.p) for _ in range(StockExponent)]
            StockDivisor= Polynomial(StockZero +[remainder.leadingCoefficient() / divisorLC], self.p)
            quotient = quotient + StockDivisor
            remainder = remainder - (StockDivisor * divisor)
        return quotient, remainder
    #modular function for polynomials
    def __mod__(self, divisor):
        x,y = divmod(self, divisor)
        return y
    def __pow__(self, p):
        x = self
        r = Polynomial(1,self.p)
        while p != 0:
            if p % 2 == 1:
                r = r * x
                p = p - 1
            x = x * x
            p = p / 2
        return r
    #polynomial to the power p modulo other
    def powmod(self, p, other):
        x,y = divmod(self**p, other)
        return y
    #usual division
    def __truediv__(self, divisor):
        if divisor.isZero():
            raise ZeroDivisionError
        x,y = divmod(self, divisor)
        return x
    #usual division
    def __div__(self, other):
        return self.__truediv__(other)
#returns a Zero polynomial
def Zero(p):
    return Polynomial([],p)
#check whether a polynomial is irreducible or not
def isIrreducible(polynomial, p):
    #polynomial "x"
    x = Polynomial([ModP(0,p), ModP(1,p)],p)
    powerTerm = x
    isUnit = lambda p: p.degree() == 0;
    for _ in range( int(polynomial.degree() / 2)):
        powerTerm = powerTerm.powmod(p, polynomial)
        gcdOverZmodp = gcd(polynomial, powerTerm - x)
        if not isUnit(gcdOverZmodp):
            return False
    return True
```

## 4. RESULTS

And the expected result is as follows.




Below table displays how efficient our proposed CLS is comparitively with

| Scheme | AP[2] | LCS[4] | YHG[7] | GS[6] | Our CLS |
|---|---|---|---|---|---|
| Sign | $3s+1p$ | $2s$ | $2s$ | $2s$ | $1s$ |
| Verify | $1e+4p$ | $2s+4p$ | $2p+2s$ | $1s+3p$ | $1s+1p$ |
| P-K-S (bits) | 320 | 320 | 160 | 160 | 160 |
| S-S (bits) | 320 | 320 | 320 | 320 | 160 |

## 5. CONCLUSION

Another worldview that rearranges the customary PKC and takes care of the inborn key escrow issue endured by ID-based cryptography is Certificateless public key cryptoanalysis. Certificateless signature is a standout amongst the most vital security primitives in CLPKC.in the irregular prophet demonstrate under the hardness presumption of k-CAA and Inv-CDHP we think of a short CLS conspire that is turned out to be secure which we proposed in this paper. Our plan, other than maintaining all attractive properties of past CLS plans, it is quicker and shorter than all proposed CLS plans as for the calculation cost and the mark measure.

## REFERENCES


[1] Al-Riyami S.S., Paterson K.G. (2003) Certificateless public key cryptography. In Proceedings of the ASIACRYPT 2003, pages 452–473. Springer-Verlag, LNCS 2894.

[2] Bellare M., Namprempre C., Neven G. (2004) Security proofs for identity-based identification and signature schemes. In: Proceedings of the EUROCRYPT 2004, p268–286. LNCS 3027. Springer-Verlag, Berlin (Full paper is available at Bellare's homepage URL: http://www-cse. ucsd.edu/users/mihir).

[3] Bellare M., Rogaway P. (1993) Random oracles are practical: A paradigm for designing efficient protocols. In: First ACM Conference on Computer and Communications Security. ACM Fairfax, pp62–73

[4] Blake-Wilson S., Menezes A. (1999) Unknown key-share attacks on the station-to-station (STS) protocol. In: Public key cryptography, second international workshop on practice and theory in public key cryptography, PKC '99. LNCS 1560. Springer-Verlag, Berlin, pp154–170

[5] Boneh D., Franklin M. (2001) Identity-based encryption from the Weil pairing. In: Proceedings of the CRYPTO 2001, LNCS 2139. Springer-Verlag, Berlin, p213–229

[6] ElGamal T. (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inform Theory 31(4):469–472 MATHCrossRefMathSciNetGoogle Scholar

[7] Girault M. (1991) Self-certified public keys. In: Proceedings of the EUROCRYPT 91, LNCS 547. Springer-Verlag, Berlin, p490–497

[8] Goldwasser S., Micali S., Rivest R. (1998) A digital signature scheme secure against adaptive chosen-message attack. SIAM J Comput 17(2):281–308 CrossRefMathSciNetGoogle Scholar

[9] Hu B.C., Wong D.S., Zhang Z., Deng X. (2006) Key replacement attack against a generic construction of certificateless signature. In: Information security and privacy: 11th Australasian conference, ACISP 2006, LNCS 4058. Springer-Verlag, Berlin, pp235–246

[10] Huang X., Susilo W., Mu Y., Zhang F. (2005) On the security of certificateless signature schemes from Asiacrypt 2003. In: Cryptology and network security, 4th international conference, CANS 2005, LNCS 3810. Springer-Verlag, Berlin, pp13–25

[11] Pointcheval D., Stern J. (1996) Security proofs for signature schemes. In: Proceedings of the EUROCRYPT 96, LNCS 1070. pp387–398

[12] Shamir A. (1984) Identity-based cryptosystems and signature schemes. In: Procedings of the CRYPTO 84, LNCS 196. Springer, Berlin, pp47–53

[13] Yum D.H., Lee P.J. (2004) Generic construction of certificateless signature. In: Information security and privacy: 9th Australasian Conference, ACISP 2004, LNCS 3108. Springer-Verlag, Berlin, pp200–211

[14] Zhang F., Safavi-Naini R., Susilo W. (2004) An efficient signature scheme from bilinear pairings and its applications. In: Seventh international workshop on theory and practice in public key cryptography (PKC 2004), LNCS 2947. Springer, Berlin, pp277–290

[15] Zhang Z., Wong D., Xu J., Feng D. (2006) Certificateless public-key signature: Security model and efficient construction. In: Fourth international conference on applied cryptography and network security (ACNS 2006), LNCS 3989. Springer, Berlin, pp293–308.