<https://eprint.iacr.org/2006/296.pdf>

2.2 identity based signature (IBS)

他是一個algorithm S=(KG,EXT,SIGN,VFY) 前三個(keygen, key derivation, sign)可以是randomize的但最後一個verify不行。

KG: input1^k安全參數放進keygen生成(PK,SK) master key和secret key

EXT: derivation algorithm 輸入SK和id去生成user secret key USK[id]

id是identity {0,1}\* 是一個01的字串

SIGN: 用USK[id]和message m簽出SIG

VFY: 用PK,id,m,SIG做verify ， return 0或1代表驗證失敗或成功

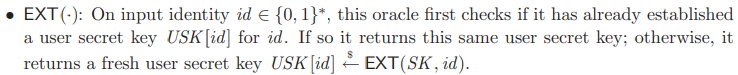
Correctness: VFY(PK, id, m, SIGN(USK [id], m) ) = 1

對應到 pk, id, m, sign

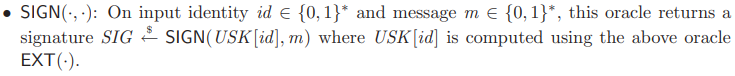
用usk[id]和m簽出來的簽名要能符合verify

安全性是在choosen-message and chosen-identity attacks下

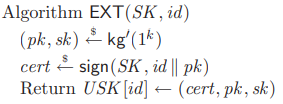
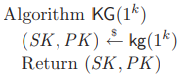
//再回顧一次和一些細節:



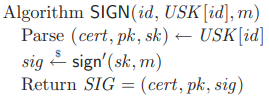
EXT是根據user給USK[id]，如果同一個user給過就會給出同一個私鑰，新的user就會給新的



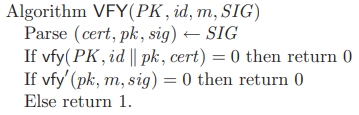
Sign是拿前面用EXT生成的user secret key USK[id]和message m生成簽章SIG



Cert是EXT時簽出來的東西 會在sign的時候用到，也會是簽名的一部分



Parse可以看成是解析，可以從USK[id]中分解出cert,pk,sk，也就是公鑰私鑰和前面的部分簽名，再拿其中的私鑰和message做sign，最後的簽章是cert pk和sig的結合



分解簽章SIG出來的參數代入verify看看是不是正確的

一些觀察: 同一個系統下同一個user的pk sk USK[id]和cert是固定的會變動的是m導致不同簽名SIG

2.3 Generic Construction of Identity-based Signatures

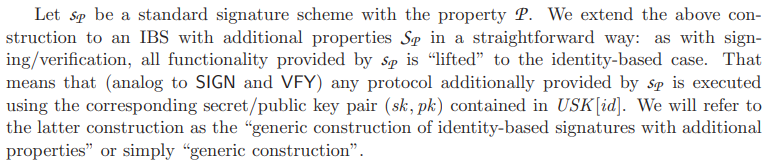
怎麼把兩個signature schemes (s,s’)結合成一個identity-based signature S ?

sp有一些additional properties P，我們有一個(similar)generic transformation可以結合s和sp來得到identity signature scheme Sp，Sp會有和sp一樣的additional properties。





如果s,s’都是安全的standard signature scheme則S=SS-2-IBS(s,s’)也是一個secure identity-based signature scheme



sP是有附加性質P的signature scheme，可以用上面的方法擴展成有附加屬性的IBS，所以sP提供的功能都可以提升到identity-based中，也就是sP提供的任何協議都可以用USK[id]中包含對應的pk,sk來執行。這種構造被稱為

generic construction of identity-based signature with additional properties或者說generic construction。

4.1 Verifiably Encrypted Signatures (VES)

簽名者可以透過adjudicator(仲裁者)的public key去創建加密簽名(VES Signature)，這簽名可以被公開驗證是否有效。Adjudicator是被信任的第三方，需要的時候會提供standard signature。VES提供了公平性，常用在簽合約

VES可以基於trapdoor function建構

Identity-based verifiably encrypted signature (IB-VES)

一般的VES scheme比standard多了三個algorithm: asign, avfy, adj。

Adjudicator的公鑰(apk)用在簽名和驗證。

私鑰(ask)用在adj algorithm，可以把VES signature轉回standard signature

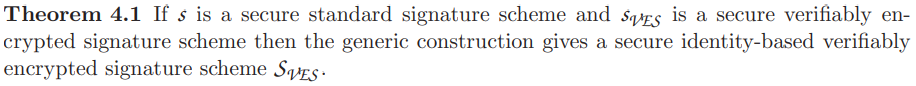


前四個算法基本和前面相同，只有KG多生成adjudicator key-pair (ask,apk)

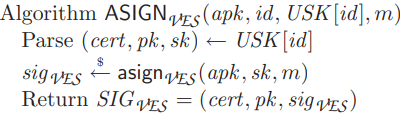
IB-VES中

1. asignVES的sig替換成VES的版本( sigVES)，是透過VES signing algorithm asign來的 (使用sk, m, apk)(簽名者的私鑰 訊息和adj公鑰)

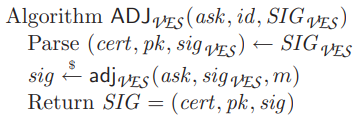
2. IB-VES verification首先檢查certificate，然後用standard verification algorithm avfy檢查。



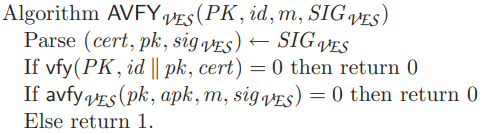
可以把安全的standard signature s和安全的VES SVES結合成安全的IBVES (SVES)



1.從USK[id]解析出cert, pk, sk 2.VES的簽名(用apk, sk, m)簽出sig 3.SIGVES包含cert, pk, sigVES



1. 從簽名SIGVES解析出cert, pk, sigVES 2.透過仲裁者私鑰ask 簽名sigVES和訊息m用adj函式轉成satndard signature sig 3.standard簽名包含cert pk sig



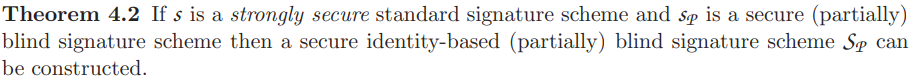
1. 從SIGVES解析出cert, pk, sigVES 2.驗證兩個公鑰PK, pk, id, cert是否正確 2.用apk驗證sigVES

4.2 (Partially) Blind Signature

盲簽章blind signature的概念是，user可以要求signer在不知道訊息m的內容時簽名，signer完全不知道m的information。

部分盲簽章Partially blind signature是signer可以在簽章中加入一些資訊，最後的receiver和signer的共通訊息

身分基盲簽章identity-based blind signature(IB-BS)，用bilinear pairings(雙線性配對)實現



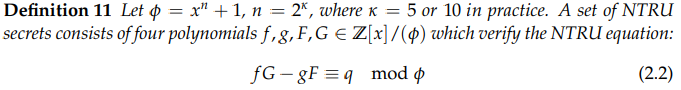
可以用一個安全的standard signature s和安全的(partially) blind signature sp

建造一個安全的identity-based (partially) blind signature scheme Sp

Dilithium論文的2.5

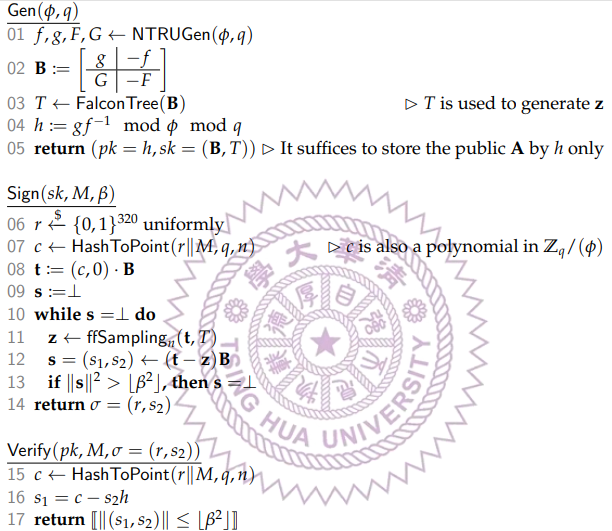
Falcon 基於SIS+ISIS 跟Dilithium比 他沒有用Fiat-Shamir technique

他是一個hash and sign scheme用NTRU lattice來建造GPV framework



NTRU格 Φ=X^n +1 ， n通常有固定，一個NTRU有4個secret polynomials f,g,F,G滿足NTRU equation

Falcon演算法



KeyGen(Φ,q)

Pk=h=g\*(f^-1) , sk=B=四個secret polynomial (g, G, f, F)的結合 和B帶入Falcon tree得到的T TREE

Sign(sk=(B,T),M, β)

可以看成r, c都是random出來的

t用random出來的c和sk的B生出來

10開始試簽章

z從前面的t和sk的T sample出來

簽名s = (s1,s2)從 (t-z)B生出來

s大小不合格就重簽

最後的簽名包括最早隨機生成的r和最後簽的s2

Verify(pk=h, M, 簽章=(r,s2) )

先利用r,M,q,n的hash做出剛才的c

做s1=c-s2h

檢查(s1,s2)大小是否符合

Falcon

Gen(Φ,q)

01 f,g,F,G <-NTRUGen(Φ,q)

02 B:=[g|-f

G|-F]

03 T<-FalconTree(B)

04 h:=g(f^-1) mod Φ mod q

05 return(pk=h,sk=(B,T))

Sign(sk,M, β)

06 r<-{0,1}^320 uniformly

07 c<-HashToPoint(r|M,q,n)

08 t:=(c,0)\*B

09 s:=⊥

10 while s=⊥ do

11 z<-ffSampling(t,T)

12 s=(s1,s2)<-(t-z)B

13 if ||s||^2 > ||β||^2 ,then s=⊥

14 return sign=(r,s2)

Verify(pk,M,sign=(r,s2))

15 c<-HashToPoint(r||M,q,n)

16 s1=c-s2\*h

17 return [||(s1,s2)||<= ||β^2||]

Dilithium

Gen

01 A<-R^(n\*m)\_q

02 (s1,s2)<-S^m\_p \* S^n\_p

03 t:=As1+s2

04 return (pk=(A,t) , sk=(s1,s2))

Sign(sk,M)

05 z:= ⊥

06 while z=⊥ do

07 y<-S^m\_r1-1

08 w1:=HighBits(Ay,2r2)

09 c from Bt:=H(M||w1)

10 z:=y+cs1

11 If ||z||>=r1-β or ||LowBits(Ay-cs2,2r2)||>=r2-β, then z:= ⊥

12 Return sign=(z,c)

Verify(pk, M, sign=(z,c))

13 w1’=HighBits(Az-ct,2r2)

14 return [||z||<r1-β] and [c=H(M||w1’)]