SIGNCRYPTION BASED ON DIFFERENT DIGITAL SIGNATURE SCHEMES

Adrian Atanasiu[1]
Laura Savu[2]

Abstract

This article presents two new signcryption schemes. The first one is based on Schnorr digital signature algorithm and the second one is using Proxy Signature scheme introduced by Mambo. Schnorr Signcryption has been implemented in a program and here are provided the steps of the algorithm, the results and some examples. The Mambo's Proxy Signature is adapted for Shortened Digital Signature Standard, being part of a new Proxy Signcryption scheme.

Keywords: signcryption; Schnorr; encryption; digital signature; Mambo, proxy.

What Is Signcryption

Signcryption is the primitive that has been proposed by Youliang Zheng in 1997 and combines public key encryption with digital signature in a single logical step, obtaining a less cost for both communication and computation [1].

Data confidentiality and data integrity are two of the most important functions of modern cryptography. Confidentiality can be achieved using encryption algorithms or ciphers, whereas integrity can be provided by the use of authentication techniques. Encryption algorithms fall into one of two broad groups: private key encryption and public key encryption. Likewise, authentication techniques can be categorized by private key authentication algorithms and public key digital signatures.

While both private key encryption and private key authentication admit very fast computation with minimal message expansion, public key encryption and digital signatures generally require heavy computation, such as exponentiations involving very large integers, together with message expansion proportional to security parameters (such as the size of a large composite integer or the size of a large finite field).

Signcryption has the intention that the primitive should satisfy "Cost(Signature & Encryption) << Cost(Signature) + Cost(Encryption)." This inequality can be interpreted in a number of ways:
• A signcryption scheme should be more computationally efficient than a native combination of public-key encryption and digital signatures.
• A signcryption scheme should produce a signcryption "ciphertext" which is shorter than a native combination of a public-key encryption ciphertext and a digital signature.

[1] Professor, Ph.D., Faculty of Mathematics and Computer Science, Bucharest University
Str. Academiei 14, Bucharest 010014, Romania, E-mail: aadrian@gmail.com,
[2] PhD Candidate, Faculty of Mathematics and Computer Science, Bucharest University
Str. Academiei 14, Bucharest 010014, Romania, E-mail: laura.savu@microsoft.com

• A signcryption scheme should provide greater security guarantees and/or greater functionality than a native combination of public-key encryption and digital signatures.

A signcryption scheme typically consists of five algorithms, Setup, KeyGenS, KeyGenR, Signcrypt, Unsigncrypt:

Setup - takes as input a security parameter 1^ k and outputs any common parameters param required by the signcryption schemes. This may include the security parameter 1^ k , the description of a group G and a generator g for that group, choices for hash functions or symmetric encryption schemes, etc.

Key Generation S(Gen) - generates a pair of keys for the sender
Key Generation R(Gen) - generates a pair of keys for the receiver
Signcryption (SC) - is a probabilistic algorithm
Unsigncryption (USC) - is a deterministic algorithm.

A signcryption scheme is a combination between a public key encryption algorithm and a digital signature scheme. A public key encryption scheme consists of three polynomial-time algorithms (EncKeyGen, Encrypt, Decrypt).

EncKeyGen - Key generation is a probabilistic algorithm that takes as input a security parameter 1^k and outputs a key pair (skenc, pkenc), written (skenc, pkenc)R←EncKeyGen(1^k). The public encryption key pkenc is widely distributed, while the private decryption key skenc should be kept secret. The public key defines a message m ∈ M and a ciphertext ∈ C.
Encrypt - Encryption is a probabilistic algorithm that takes a message m ∈ M and the public key pkenc as input and outputs a ciphertext C ∈ C, written C ←Encrypt(pkenc,m)
Decrypt - Decryption is a deterministic algorithm that takes a ciphertext C ∈ C and the private key skenc as input and outputs either a message m ∈M or the failure symbol ⊥, written m ← Decrypt(skenc,C).

On the base of the first scheme that I present in this paper stands the Schnorr digital signature. A Schnorr signature is a digital signature produced by the Schnorr signature algorithm. Its security is based on the intractability of certain discrete logarithm problems. It is considered the simplest digital signature scheme to be provably secure in a random oracle model. It is efficient and generates short signatures.

Elgamal Signcryption

Based on discrete algorithm problem, Signcryption cost is:
58% less in average computation time
70% less in message expansion
Signcryption parameters:
p = a large prime number, public to all
q = a large prime factor of p-1, public to all
g = an integer with order q modulo p, in [1,… , p-1], public to all
hash = a one-way hash function
KH = a keyed one-way hash function = KHk(m) = hash(k, m)

(E, D) = the algorithms which are used for encryption and decryption of a private key cipher.
Alice sends a message to Bob.
Alice has the pair of keys (Xa, Ya):
Xa = Alice's private key, chosen randomly from [1, .., q-1]

Ya = Alice's public key = g^{Xa} mod p
Bob has the pair of keys(Xb, Yb):
Xb = Bob's private key, chosen randomly from [1, .., q-1]
Yb = Bob's public key = g^{Xb} mod p.
In order to signcrypt a message m to Bob, Alice has to accomplish the following operations:
Calculate

$k = hash(Yb^X) \ mod \ p$
Split k in k1 and k2 of appropriate length.
Calculate r = KHk2(m) = hash(h2, m)
Calculate s = x/(r+Xa) mod q, if SDSS1 is used
Calculate s = x/(1+Xa · r) mod q, if SDSS2 is used
Calculate c = Ek1(m) = the encryption of the message m with the key k1.
Alice sends to Bob the values (r, s, c).
In order to unsigncrypt a message from Alice, Bob has to accomplish the following operations:
Calculate k using r, s, g, p, Ya and Xb
$\text{hash}((Ya \cdot g^r)^{s \cdot Xb} \bmod p)$, if is used SDSS1
$\text{hash}((g \cdot Ya^r)^{s \cdot Xb} \bmod p)$, if is used SDSS2
Split k in k1 and k2 of appropriate length.
Calculate m using the decryption algorithm m = Dk1(c).
Accept m as a valid message only if KHk2(m) = r.
Using the two schemes SDSS1 and SDSS2, two signcryption schemes have been created, SCS1 and SCS2, respectively. The two signcryption schemes share the same communication overhead, (|hash(*)| + |q|). SCS1 involves one less modular multiplication in signcryption then SCS2, both have a similar computational cost for unsigncryption [1].

Schnorr Signcryption

Schnorr Digital Signature Algorithm

A Schnorr signature is a digital signature produced by the Schnorr signature algorithm. Its security is based on the intractability of certain discrete logarithm problems. It is considered the simplest digital signature scheme to be provably secure in a random oracle model [3].

Choosing parameters
All users of the signature scheme agree on a group G with generator g of prime order q in which the discrete log problem is hard.

Key generation

Choose a private signing key x.
The public verification key is y = gx.

Signing
To sign a message M:
Choose a random k.
Let r = gk
Let e = H(M | | r), where || denotes concatenation and r is represented as a bit string. H is a cryptographic hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$.
Let s = (k − xe).
The signature is the pair (s,e).

Verifying
Let rv = gsye
Let ev = H(M | | rv)
If ev = e then the signature is verified.

Demonstration of correctness
It can be observed that ev = e if the signed message equals the verified message:
rv = gsye = gk − xegxe = gk = r, and hence ev = H(M | | rv) = H(M | | r) = e.
It has been considered that k < q and the assumption that the hash function is collision-resistant.
Public elements: G, g, q, y, s, e, r.
Private elements: k, x. [4]

Schnorr Signcryption Algorithm

Schnorr Signcryption parameters:

p = a large prime number, public to all
q = a large prime factor of p-1, public to all
g = an integer with order q modulo p, in [1,… , p-1], public to all
hash = a one-way hash function
KH = a keyed one-way hash function = KHk(m) = hash(k, m)
(E, D) = the algorithms which are used for encryption and decryption of a private key cipher.
Alice sends a message to Bob.
Alice has the pair of keys (Xa, Ya):
Xa = Alice's private key, chosen randomly from [1, .., q-1]
Ya = Alice's public key = g^{-Xa} mod p
Bob has the pair of keys(Xb, Yb):
Xb = Bob's private key, chosen randomly from [1, .., q-1]
Yb = Bob's public key = g^{-Xb} mod p.
In order to signcrypt a message m to Bob, Alice has to accomplish the following operations:

Calculate

$k = hash(Yb^X) \bmod p$
Split k in k1 and k2 of appropriate length.
Calculate r = KHk2(m) = hash(h2, m)
Calculate s = x + (r * Xa) mod q
Calculate c = Ek1(m) = the encryption of the message m with the key k1.
Alice sends to Bob the values (r, s, c).
In order to unsigncrypt a message from Alice, Bob has to accomplish the following operations:
Calculate k using r, s, g, p, Ya and Xb
$\boldsymbol{k = hash\left((g^s * Ya^r)^{-Xb} \bmod p\right)}$
Split k in k1 and k2 of appropriate length.
Calculate m using the decryption algorithm m = Dk1(c).
Accept m as a valid message only if KHk2(m) = r.

Implementation Of Schnorr Algorithm

Calculare Ya and Yb

```
double powA = Math.Pow(g, xA);
int pow_intA = Convert.ToInt32(powA);
int invA = modInverse(pow_intA, p);
```

Calculate k

```
int yB = Convert.ToInt32(textBox11.Text);
int x = Convert.ToInt32(textBox18.Text);
int p = Convert.ToInt32(textBox4.Text);
string cheie = (BigInteger.ModPow(yB, x, p)).ToString();
```

Calculate hash(k)

```
string HashDeCheie = _calculateHash(cheie);
textBox13.Text = HashDeCheie;
```

Split k in two keys k1 and k2 with the same lenght

```
byte[] k = Convert.FromBase64String(textBox13.Text);
byte[] k1 = new byte[k.Length/2];
byte[] k2 = new byte[k.Length - k.Length / 2];
            Buffer.BlockCopy(k, 0, k1, 0, k.Length/2);
Buffer.BlockCopy(k, k.Length / 2, k2, 0, k.Length - k.Length / 2);
byte[] test = new byte[k.Length];
k1.CopyTo(test, 0);
k2.CopyTo(test, k1.Length);
```

Calculate r using k2; r = hash (k2, m)

```
BigInteger p = BigInteger.Parse(textBox4.Text);
System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
byte[] keyByte = encoding.GetBytes(key);
HMACSHA1 hmacsha1 = new HMACSHA1(keyByte);
byte[] messageBytes =encoding.GetBytes(message);
byte[] hashmessage = hmacsha1.ComputeHash(messageBytes);
```

Calculate r using k2; transform the value obtained from hash in base 10

```
textBox19.Text = fn16to10(textBox15.Text).ToIntString();
```

Calculate the modulo p of the number obtained in base 10

```
BigInteger nr=BigInteger.Parse(textBox19.Text);
BigInteger p = BigInteger.Parse(textBox4.Text);
BigInteger  rest = 0;
BigInteger.DivRem(nr, p, out rest);
```

Calculate s

```
BigInteger q = Convert.ToInt32(textBox5.Text);
BigInteger  r = Convert.ToInt32(textBox20.Text);
BigInteger XA = Convert.ToInt32(textBox9.Text);
BigInteger X = Convert.ToInt32(textBox18.Text);
BigInteger prod = BigInteger.Multiply(r, XA);
BigInteger sum = X + prod;
BigInteger rest;
BigInteger.DivRem(sum, q, out rest);
```

Encrypt m using the k1
Calculate k

```
BigInteger rez2 = BigInteger.Pow(rez1, XB);
BigInteger invK = modInverseBI(rez2, p);
```

Example:
p = 23, q = 11, g = 2, X=3
XA=4 => YA=13
XB=5 => YB=18
k = 13 => hash(k) = vTB6PsMp4Qos/4+4dICCPaEU+PQ=
k1 = vTB6PsMp4Qos/w==
k2 = j7h0gII9oRT49A==
hash(k2, m) = E2726583242AB5CCE58AE1151DB126208F17932F
hash(k2,m) in base 10 = 1292783042124763369608714420962730428414981280559
(hash(k2,m) in base 10) mod p = 3
s mod q = x+(r*Xa) mod q = 4
Unsigncrypt k = 13

A New Proxy Signcryption Scheme

The implementation of the initial signcryption cryptographic primitive is built up on the ElGamal signature scheme variations. The two most important ElGamal digital signature variations are Schnorr Signature and Digital Signature Standard (DSS). For the signature standard were created two shorter algorithms, SDSS1 and SDSS2.

In the following section it is presented the Mambo's proxy signature scheme adapted to the Shorten Digital Signature Standard 2 (SDSS2).

Signature Phase:
x = a secret random number from [1,..q-1] which is different for each signing operation of Alice
r = hash(g^x mod p, m)

s = $\frac{x}{1+r*Xa}$ mod q
Alice's signature on a message m is sig_a(m) = (r, s).

Verification Phase:
Calculate k =$(g * Ya^r)^s \ mod\ p$.
Bob is using his private key and the public parameters in the above formula in order to calculate k.
The proxy agent receives a proxy secret from Alice.
This is (x_{ap} , k) where:
x = a secret random number from [1, .., q-1] chosen specifically for creating the proxy key.
k = g^x mod p and
x_{ap} = x_a + x * k mod p-1
Alice provides the pair (x_{ap}, k) to the proxy agent.
The proxy agent makes the verification.
The proxy agents calculates k_p = $g^{x_{ap}}$ mod p.
It is calculates $k_p$using the following formula

$$k_p = \left(Ya * k^k \right) mod\ p$$

x_{ap} is accepted as a valid proxy from Alice ⇔ k_p = $g^{x_{ap}}$.
The difficulty of the Discret Logarithm Problem (DLP) keeps the value of x_a as a secret for the proxy agent.
x_{ap} is the proxy secret key used by the proxy agent to sign a message m on behalf of Alice. Bob receives from the proxy agent the signed message (m, $sig_{ap}(m), k$) and verifies its origin using the public proxy key, y_{ap} = y_a * k^k mod p.

Signing using proxy signature for SDSS2
Parameters used by the proxy signature:
x' = a secret random number from [1, ..q-1] chosen independently for each signing operation by proxy agent
r' = hash($g^{x'}$ mod p, m)
s' = $\frac{x'}{1+r'*Xap}$ mod q
The proxy signature of Alice on the message m is the following: $sig_{ap}(m) = (r', s', k)$

Verification using proxy signature for SDSS2
Bob uses the received values and the public parameters in order to calculate k from the following formula:
k = $(g * {y_{ap}}^{r'})^{s'}$ mod p.
Bob accepts m as a valid message from Alice ⇔
hash (k, m) = r'.

A New Proxy Signcryption Scheme

In this section is presented the algorithm of the new proxy signcryption scheme. A proxy signcryption scheme is a primitive that combines the public key encryption with the proxy

signature with the scope to obtain less computational and communicational costs comparing with the method “proxy signature then encryption”.

Setup

Alice creates the proxy secret key (x_{ap}, K) and transmit it securely to the trusted proxy agent.

Parameters involved:

x = a secret random number from [1,..q-1] which is chosen specifically for creating the proxy right.

K = g^x mod p and

$x_{ap} = x_a + x * K \mod p-1$

$y_{ap} = y_a * K^K \mod p$ is the public proxy key.

Sign

Here are listed the parameters used in signcryption by Alice proxy using the proxy signature scheme.

x’ = a secret random number from [1, ..q-1] chosen independently for each signing operation by the proxy agent

k = ${y_b}^{x'} \mod p$

Split k in k_1 and k_2 with k = $k_1 || k_2$

r’ = $KH_{k_2}(m) = hash\ (k_2, m)$

s’ = $\frac{x'}{1 + r' * x_{ap}} \mod q$

Encrypt the message m with the encryption algorithm E and the key k_1.

c = $E_{k_1}\ (m)$

The final result of the proxy signcryption process of Alice on the message m with the proxy signature is represented by

$cryptogram_{ap}$ (m) = (c, r’, s’, K).

Verification

Bob receives from Alice (c, r’, s’, K).

Bob recovers K from signature (r’, s’, K) and using the public parameters (g, p, y_a) and his secret key x_b, he is able to calculate k using the following formula:

k = $(g * {y_{ap}}^{r'})^{s' * x_b} \mod p$.

When he has the value of k, he splits it in k_1 and k_2 so that

k = $k_1 || k_2$.

Now Bob can recover the message m using decryption algorithm D and the key k_1

m = D_{k_1} (c)

m is accepted as the valid message from Alice ⇔

r’ = $KH_{k_2}(m) = hash\ (k_2, m)$.

Proxy Signcryption Security

The strength of the proxy signature scheme on which we based our implementation is not greater than the strength of the underlaying ordinary signature scheme. For the proxy signcryption scheme described above, the cryptographic strength is the same as for the

general signcryption primitive which is turn is based on the intractability of the discret logarithm problem [9]. The proxy signature scheme has several proprieties that make it resistant to a range of attacks.

As Alice's secret key $x_a$cannot be computed from the proxy secret (x_{ap}, K) and public known parameters, a corrupt proxy agent cannot forge delegation of signature rights by itself. Also, a malicious third-party that breaks-in to the principal to steal the secret x_a cannot duplicate a previously generated proxy secret without knowing the random value x used in the key generation. Therefor a proxy signcryption by a proxy agent is unforgeable except in the case of a malicious principal [9].

As the signature verification is distinctly different for an original signature and a proxy signature, a corrupt proxy agent cannot deny authorship of a signature without claiming complete loss of its proxy secret. An important issue of the proxy signatures is the revocation of a proxy right granted to a proxy agent. A straightforward mechanism is to revoke the principal's public key, thus preventing future acceptance of proxy signatures by recipients as they can no longer be correctly verified [9].

Conclusions

This paper presents two new Signcryption schemes. The first one is based on Schnorr digital signature algorithm and the second one uses in its construction the proxy signature scheme. For Schnorr Signcryption is presented the step-by-step source code implementation algorithm. The second signcryption scheme combines the public key encryption with the proxy signature scheme which has been introduced for the first time by Mambo [10], [11]. A similar proxy signcryption was proposed before in literature but it was created on the Shorten Digital Signature Standard 1, SDSS1. In the construction of the initial signcryption primitive, both Shorten Digital Signature Standards have presented.

References

1. Y. Zheng. Digital signcryption or how to achieve cost(signature & encryption) << cost(signature) + cost(encryption). Full version. Available from http://www.sis.uncc.edu/yzheng/papers/ , 1997.
2. Alex Dent and Yuliang Zheng: Practical Signcryption, a volume in Information Security and Cryptography, Springer-Verlag, Berlin, November 2010. (ISBN: 978-3-540-89409-4)
3. C.P. Schnorr, Efficient identification and signatures for smart cards, in G. Brassard, ed. Advances in Cryptology—Crypto '89, 239-252, Springer-Verlag, 1990. Lecture Notes in Computer Science, nr 435.
4. Claus-Peter Schnorr, Efficient Signature Generation by Smart Cards, J. Cryptology 4(3), pp161–174 (1991).
5. J. H. An, Y. Dodis, and T. Rabin. On the security of joint signatures and encryption. In L. Knudsen, editor, Advances in Cryptology – Eurocrypt 2002, volume 2332 of Lecture Notes in Computer Science, pages 83–107. Springer, 2002.
6. R. Steinfeld and Y. Zheng. A signcryption scheme based on integer factorization. In J. Pieprzyk, E. Okamoto, and J. Seberry, editors, Information Security Workshop (ISW 2000), volume 1975 of Lecture Notes in Computer Science, pages 308–322. Springer, 2000.

7. International Organization for Standardization. ISO/IEC WD 29150, IT security techniques — Signcryption, 2008.
8. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, Advances in Cryptology – Asiacrypt 2000, volume 1976 of Lecture Notes in Computer Science, pages 531–545. Springer, 2000.
9. Chandana Gamage , Jussipekka Leiwo , Yuliang Zheng, "An Efficient Scheme for Secure Message Transmission using Proxy-Signcryption".
10. M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures: Delegation of the power to sign messages. IEICE Trans. Fundamentals Sep. 1996, Vol. E79-A, No. 9, pp. 1338-1353.
11. M. Mambo, K. Usuda, E. Okamoto. Proxy signatures for delegating signing operation. In: *3rd ACM Conference on Computer and Communications Security* (*CCS'96*), pp. 48-57. New York: ACM Press, 1996.