DAT 510: Assignment 2

Tomasz Gliniecki

5.11.2016

Abstract

This project focuses on digital signature standards. Over the years there has been couple proposals. This solution implements DSA, current Digital Signature Standard (DSS). Reason why we use digital signature is to be able to ”sign”, digital files in order to confirm that certain file comes from the sender we are expecting. Digital signature is used to provide authenticity, data integrity and non-repudiation. Digital signature does not provide confidentiality in it self, the usecase for digital signature is very different form normal encryption, even though there is some encryption involved in digital signature, it is privet key encryption. As we know, if we assume that only the owner of private key knows the private key, we can use it to create a signarute, as no other party will be able to reproduce the steps we make diring signing withouth the private key. The receiver of the signature can confirm the signature by using senders public key in order to confirm authenticity.

Introduction

DSA algorithm was combined together with SHA512, in order to create a working prototype of digital signature scheme. The program that was written is only usefoul for testing purposes, but the mechanismsand logic inside are supposed to be on the level with publicaly available free implementations. The general overview of the program is very straighforward, cryptographic algorithms do not leave much room for playing with different components, because it is very difficult to predict the security vulnerabilities that might arise by jsut chaning one simple thing in the algorithm. The only ”open” module in DSS is the hashing algorithm, it is always advized to use an approved cryptographic hash function, that is secure. Usually SHA-1 was used in that place, but we are slowly moving towards more secure hashing algorithms like SHA-2. As mentioned earlier, we do not care about the whole message being unencrypted, the purpose of digital signature is not that, on the other hand we simply encrypt the calculated hash of the file we want to send. This is becasue encrypting the whole message with the same method would take very long time, we would rather calculate secure hash that by definition has a fixed length, and simply encrypt that hash with our private key. This result together with unencrypted message/file concatinated would be sent to the recipient. On the other side, the rcipient sees the message in plain form, and calculates the hash of that message, and compares this with the output of decryption of whateevr he recieved together with the plain message. This is the basig flow of digita signature. DSS algorithm is a bit more complicated than that.

Design and Implementation

Implementation of SHA512 is very straightforward, it involves very little complex calculations, the only difficult part is to implement correctly and following the steps of algorithm religiously. SHA hashing family is very similar, the most important bit that changed in different version is the block size and output size. Most of the logic is the same, but some amounts change better overview can be found here [1].
DSA (Digital Signature Algorithm) - one of the the most interesting algorithms used standard digital signature (Digital Signature Standard DSS).
Cryptosystem was developed by the NSA (National Security Agency) and was approved as a standard digital signature in the US in 1999.
The principle of the algorithm is based on DSA following parameters:
p prime number where $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L a multiple of 64; i.e., bit length of between 512 and 1024 bitsin increments of64 bits.
q prime divisor of $(p-1)$, where $2^{159} < q < 2^{160}$

$g = h^{(p-1)/q} mod p$, where h is any integer with 1 ¡ h ¡ (p-1) such that $h^{(p-1)/q} mod p > 1$

Users private Key

$x-$ random or pseudorandom integer with 0 ¡ x ¡ q

Users publick Kye

$y = g^x mod p$

Users's per-message secret number

$k = random or pseudorandom integer with 0 < k < q$

Signing :

$r = (g^k mod p)\ mod q$

$s = [k^{-1}(H(M) + xr)]mod\ q$

Verifying:

$w = (s')^{-1} mod\ q$

$u_1 = [H(M')w]mod\ q$

$u_2 = (r')w mod\ q$

$v = [(g^{u1}y^{u2}) mod\ p\ ]mod q$

Then the test is passed iff $v = r'$ algorith source taken from [2]

implementation in C++ look like this (in short):

```
void Client::sign() {
  gmp_randstate_t state;gmp_randinit_mt(state);
  unsigned long int seed = time(NULL);
  gmp_randseed_ui(state, seed);
  // setting user per message secret number
  mpz_urandomm (this->pmsn, state, this->q);

  // encrpt (this->hash_of_msg) with private_key
  mpz_init(sign_r);
  mpz_init(sign_s);
  // signgning
  // r = (g^k mod p) mod q
  mpz_powm(this->sign_r, this->g, this->pmsn, this->p);
  mpz_powm_ui(this->sign_r, this->sign_r, 1, this->q);

  //  k
  mpz_t inv_pmsn; mpz_init(inv_pmsn);
  mpz_invert (inv_pmsn, this->pmsn, this->q);

  mpz_init(this->hash_of_msg);

  std::string str;
  str = sha512(this->msg_out);
  const char * c = str.c_str();
  mpz_set_str(this->hash_of_msg,c,16);
  cout << endl;
  cout << "hash of the message that will be used for signing: " << endl;
  cout << str << endl << endl;

  // calculating s = { k  1  (H(M) + xr)} mod q
  mpz_t intermidiet; mpz_init(intermidiet);
  mpz_mul(intermidiet, this->x, this->sign_r);

  mpz_add(this->sign_s, this->hash_of_msg, intermidiet);
  mpz_mod (this->sign_s, this->sign_s, this->q);

  // signature is now this->(sign_r, sign_s);

}
```

Listing 1: signing the message with DSS and sha512

```

bool Client::verify_message(Client *cli) {
  // TODO:
  // set signature_verification = H(msg_in)
  // compare D(signature_in) == signature_verification
  // in short if
  //      D(signature_in) == H(msg_in) then pass
  // test if this->in_sign_r == sign_r_ver
  mpz_init(this->sign_r_ver);
```

```
mpz_t w; mpz_init(w);

mpz_t inv_in_sign_s; mpz_init(inv_in_sign_s);
mpz_invert (inv_in_sign_s, this->in_sign_s, this
    ->q);

mpz_mod(w, inv_in_sign_s, this->q);

mpz_t u1, u2; mpz_init(u1); mpz_init(u2);

std::string str;
str = sha512(this->msg_in);
const char * c = str.c_str();
mpz_set_str(this->hash_of_msg,c,16);

mpz_mul(u1, this->hash_of_msg, w);
mpz_mod(u1, u1, this->q);

mpz_mul(u2, u2, w);
mpz_mod(u2, u2, this->q);

mpz_t g_u1, y_u2, v; mpz_init(g_u1); mpz_init(
    y_u2); mpz_init(v);
//(A * B) mod C = (A mod C * B mod C) mod C
mpz_powm(g_u1, this->g, u1, this->p);
mpz_powm(y_u2, cli->public_key, u2, this->p);
mpz_mul(v, g_u1, y_u2);
mpz_mod(v, v, this->p);
mpz_mod(v, v, this->q);

cout << "Verifying that the checksum is correct :
    " << endl;
mpz_out_str(stdout,16,cli->sign_r);cout<<endl;
mpz_out_str(stdout,16,this->in_sign_r);cout<<endl
    ;

// if v == this->in_sign_r then pass
if (mpz_cmp(cli->sign_r, this->in_sign_r) == 0) {
  return true;
} else {
  return false;
}

```

```
}
```
Listing 2: verification step of DSS with SHA512

As one can notice, this involves a lot of mathematical operation on big numbers. This is not hadr to implement, but the theory behind it is very much complicated, and it requuires at least understangin of basic modular mathematic to implement it without major problems. The implementation follows the algorithm step by step, there is not much freedom here except the programing language used and solving the big integer arithmetic problem.

0.1 Discussion

The results of running the program are as expected. The most cpu intensive work is done in the begiining of the program, men public-key components are being calculated. This step involves finding big primes over and over again to satisfy certain cryptographical requirements for cyclic groups. The good thing is that we are not required to calculate all these number everytime, once calcualted it can be stored separately and then accessed as necessary. The parts that are required to be processed and computed at more frequently are much less recource intensive than strong prime generation.

Conclusion

As always with cryptography, it is very important to keep everything up to the standards in industry, and implement functions correctly. It is very difficult to debug many cryptographical implementations, as the programs we run might work whether we are implementing security correctly or not. In the case of DSS implementation, again the most difficult part is to be able to handle big number arithmetic correctly. This is the core and essence of security in many cases. Improvements to the current solution could be plenty, try to use the solution by implementing it in a bigger program, incorporating into bigger environment and seeing how it could fit in there. What other usecases of digital signature could there be beside the standard theoretical explanations. All those different building blocks that sum up to greade digital signature are used in different arrangements and combinations to supply for different demands.

To summarize what we have learned: Digital signature is used for authentication purposes mostly, there exist many different algorithms for digital signature, as well as different hashing algorithms but only few of those can provide sufficient security to be cryptographically secure. As the technology gets better with years, we will probaly need better digital signature solutions, new lolutions liek SHA-3 are already in development for long time, and are already in use.

Works Cited

[1] Sha 512 wikipedia
`https://en.wikipedia.org/wiki/SHA-2`

[2] Digital signature standart p.405 Cryptography and Network Security book