Licence L2 STS Mention SPI Parcours Informatique
Unité 174EN007
Sécurité Informatique

Correction TD3

Ce TD porte sur différents aspects cryptologiques.

Exercice n°1 : algorithme Deffie-Helman

(a) D'après vous, est-ce que Alice et Bob ont la même valeur k ? Par conséquent, à quoi peut servir le protocole Deffie-Hellman ?

Alice et Bob partage la même valeur k car $y_2^{x1} = y_1^{x2}$ bien que cette valeur n'est pas été échangée.

Le protocole Deffie-Hellman permet de partager un secret , par exemple une clé secrète ou une valeur permettant de générer une clé secrète en appliquant une fonction génératrice.

(b) Testez l'algorithme avec $p = 23$ et $a = 3$.

1. Alice et Bob choisissent un nombre premier p et une base a. Dans notre exemple, $p = 23$ et $a = 3$.
2. Alice choisit un nombre secret $x_1 = 6$. Bob choisit à son tour un nombre secret $x_2 = 3$
3 et 4. Alice envoie à Bob la valeur $A = a^{x1}\ [mod\ p] = 3^6\ [23] = 729\ [23] = 16$
*(729 - [729/23] * 23)*
Bob envoie à Alice la valeur $B = a^{x2}\ [mod\ p] = 3^3\ [23] = 27\ [23] = 4$
(27 – 23)
5. Alice peut maintenant calculer la valeur secrète : $(B)^{x1}\ [mod\ p] = 4^6\ [23] = 2$
*(4096 – 178 * 23)*
Bob fait de même et obtient la même valeur qu'Alice : $(A)^{x2}\ [mod\ p] = 16^3\ [23] = 2$
*(4096 – 178 * 23)*

(c) D'après vous, sur quel principe mathématique se fonde le protocole Deffie-Hellman ?

Le protocole utilise l'arithmétique modulaire (calcul avec des modulos) et sur le postulat suivant :

Etant donnés des entiers p, a, x avec p premier et $1 \leq a \leq p - 1$:
- il est facile de calculer $y = a^x\ (mod\ p)$;
- connaissant y, a et p, il est très difficile de retrouver x pourvu que p soit très grand.

Retrouver x connaissant y, a et p s'appelle résoudre le problème du logarithme discret. C'est un problème mathématique pour lequel on ne dispose pas d'algorithme efficace.

Dans le cas du protocole, on connaît publiquement p, a, y_1 et y_2 mais pas x_1 ou x_2.

(d) D'après vous, ce protocole est-il vulnérable à une attaque de type homme du milieu ? Si oui proposez une solution.

Oui, si un attaquant peut se placer entre Alice et Bob, intercepter la clé y_1 envoyée par Alice et envoyer à Bob une autre clé $y_1' = a^{x1'} \pmod p$, se faisant passer pour Alice. De même, il peut remplacer la clé y_2 envoyée par Bob à Alice par une clé $y_2' = a^{x2'} \pmod p$, se faisant passer pour Bob. L'attaquant communique ainsi avec Alice en utilisant la clé partagée $y_1^{x2'}$ et communique avec Bob en utilisant la clé partagée $y_2^{x1'}$, Alice et Bob croient communiquer directement.

Alice et Bob croient ainsi avoir échangé une clé secrète alors qu'en réalité ils ont chacun échangé une clé secrète avec l'attaquant, l'homme du milieu.

Pour se prémunir de ce genre d'attaque, il faut utiliser des signatures à clés publiques, une pour Alice et une pour Bob, soit déjà partagées soit par certificat numérique.

Exercice n°2 : étude de HMAC (*keyed-Hash Message Authentification Code*) et implémentation avec SHA256.

(a) Lisez l'extrait de la RFC-2104.

Voir documentation fournie.

(b) Donnez un algorithme générique du calcul HMAC.

```
function hmac (key, message, blocksize)
   if (length(key) > blocksize) then
      key = hash(key) // keys longer than blocksize are shortened
   end if
   if (length(key) < blocksize) then
      // keys shorter than blocksize are zero-padded (where . is concatenation)
      key = key . [0x00 * (blocksize - length(key))]

   end if

   o_key_pad = [0x5c * blocksize] ⊕ key // Where blocksize is that of the underlying hash function
   i_key_pad = [0x36 * blocksize] ⊕ key // Where ⊕ is exclusive or (XOR)

   return hash(o_key_pad . hash(i_key_pad . message)) // Where . is concatenation
end
```

(c) Donnez une implémentation en langage C de HMAC-SHA256.

```
#include <string.h>
#include "hmac_sha256.h"

void hmac_sha256(BYTE text[], int text_len, BYTE key[], int key_len, BYTE hash[]) {
        SHA256_CTX ctx;

        BYTE k_ipad[64];
        BYTE k_opad[64];
        BYTE tk[32];
```

```
	int i;

	// Si la clé est plus grande que 64 octets alors key = SHA256(key)
	if (key_len > 64) {
		sha256_init(&ctx);
		sha256_compute(&ctx,key,key_len);
		sha256_convert(&ctx,tk);
		key = tk;
		key_len = 32;
	}

	// Si la clé est plus petite elle est complétée avec des zéros
	// et copiées dans k_ipad et k_opad
	memset(k_ipad, 0, sizeof k_ipad);
	memset(k_opad, 0, sizeof k_opad);
	memcpy(k_ipad, key, key_len);
	memcpy(k_opad, key, key_len);

	// key XOR ipad et key XOR opad
	for (i=0; i < 64; i++) {
		k_ipad[i] ^= 0x36;
		k_opad[i] ^= 0x5c;
	}

	// Empreinte sha256(k_ipad + text)
	sha256_init(&ctx);
	sha256_compute(&ctx, k_ipad, 64);
	sha256_compute(&ctx,text, text_len);
	sha256_final(&ctx);
	sha256_convert(&ctx, hash);

	// Empreinte sha256(k_opad + sha256(k_ipad + text))
	sha256_init(&ctx);
	sha256_compute(&ctx, k_opad, 64);
	sha256_compute(&ctx, hash, 32);
	sha256_final(&ctx);
	sha256_convert(&ctx,hash);
}
```

Exercice n°3 : le carré de Polybe et le chiffre ADFGVX

La substitution avec le carré de Polybe 6x6 donne :

AG AX GF VD AX AV GF FA GD VV GD

Le tableau de transposition est le suivant :

G	***E***	***N***	***I***	***A***	***L***
3	***2***	***6***	***4***	***1***	***5***
A	G	A	X	G	F
V	D	A	X	A	V
G	F	F	A	G	D
V	V	G	D		

Le code est : GAGGD FVAVG VXXAD FVDAA FG