Cryptographie

26 septembre 2023

On suit le introduction to modern cryptography de katz et lindell je crois. (celui que j'ai en partie lu)

1 formalisme

$\mathcal{M}$ un ensemble.

1.1 Schéma de signature numérique

Définition 1.1.1. Triplet d'algorithme probabilistes, Π,(classe avec trois élts) "efficaces"(1/10 de sec).

- KeyGen(void) : retourne (s_k, v_k), clé de signature(privée), clé de vérification(publique parfois).
- sgn Sign(s_k, m) prend clef privée et msg et renvoie une signature σ.
- bool Verify(v_k, m, σ) renvoie vrai ou faux.

Tout ça tel que $\forall\ m \in \mathcal{M}$, si $(s_k, v_k) \leftarrow KeyGen()$

$$Verify(v_k, m, Sign(s_k, m)) = 1$$

1.2 Securité (ϵ, t)

A éviter :

- Manque de formalisme.

- Avoir un modèle trop fort qui ne peut être vérifié (il y a des attaques structurelles)(Une personne a 50 km d'une centrale est une attaque)
- modèle trop faible.(Un groupe de personnes armée attaque la centrale pas considéré comme une attaque)(ne prend pas en compte des attaques réalistes)

Définition 1.2.1. Def bancale :1. Π est sécurisé si tout individu malveillant qui est pas censé pouvoir signer ne peut pas signer.

A defs "qui est pas censé pouvoir signer".

Définition 1.2.2. 2. Π est securisé si $\forall m \in \mathcal{M}$ $\forall (v_k, s_k) \leftarrow Keygen()$, il n'existe pas d'individu qui peut produire une signature σ, tel que $Verify(v_k, m, \sigma) = 1$.

Définition 1.2.3. 3. Π est securisé, si $\forall m \in \mathcal{M}$ $\forall (s_k, v_k) \leftarrow Keygen()$...

bon la vrai def :(ϵ, t) sécurité

Définition 1.2.4. Π est securisée si $\forall m \in \mathcal{M}$. Il n'existe pas d'algo A finissant en temps t tel que

$$Pr(Keygen() \to (s_k, v_k)\&\&(Verify(v_k, m, A(v_k)) == 1)) \leq \epsilon$$

On définit $ExpInforg(A, \Pi, m)$:
-$(s_k, v_k) \leftarrow \Pi.Keygen()$
-$\sigma \leftarrow A(v_k)$
-Retourne $\Pi.Verify(v_k, m, \sigma)$

On réecrit la def avec:

Définition 1.2.5.

$$Pr(ExpInforg_{A,\Pi,m}() == 1) \leq 10^{-10}$$

Un souci avec $ExpInforg$ (modèle trop faible). Si Π était sécurisé on pourrait construire Π' pareil que Π ou verify renvoie 1 aussi si $m =$ message compromettant signé par la mauvaise personne.(il est aussi sécurisé et la un message absurde est vérifié) (pas sur d'avoir compris) (en gros m est choisi en amont)
Je crois qu'on dit que on choisit plus m a l'avance c'est A qui le génère.

On redéfinit $ExpInforg_{A,\Pi}()$:
-$(s_k, v_k) \leftarrow \Pi.Keygen()$

-$(m,\sigma) \leftarrow A(v_k)$
-Retourne $\Pi.Verify(v_k, m, \sigma)$

On sait aussi que dans le monde réel l'attaquant peut avoir accès à des paires (m, σ) particulières.(Il suffit que quelqu'un signe un msg alors une paire est disponible, trivialement)

Le pb vient quand on arrive à signer un msg qui n'a pas a être signé.

On redéfinit $ExpInforgdyn_{A,\Pi}()$:
-$(s_k, v_k) \leftarrow \Pi.Keygen()$
-$Q = \emptyset$ qui stockera tout les messages signés.(variable globale)
On ajoute un algorithme (l'oracle)

- $O_{s_k}(m)$:
- $Q = \{m\} \cup Q$
- $\sigma \leftarrow \Pi.Sign(s_k, m)$
- retourne σ

-$(m,\sigma) \leftarrow A^{O_{s_k}()}(v_k)$
-Retourne $\Pi.Verify(v_k, m, \sigma) \&\& m \notin Q$

Ducoup la

Définition 1.2.6. Π est (ϵ, t)-securisé si il n'existe pas de A tournant en temps t tq

$$Pr((s_k, v_k) \leftarrow Keygen() \&\& ExpInforgdyn_{A,\Pi}() == 1) \leq \epsilon$$

1.3 Sécurité asymptotique

Définition 1.3.1. $f : \mathbb{N} \to \mathbb{R}$ est négligeable si pour tout $k \in \mathbb{N}$:

$$f(\lambda) = O(1/\lambda^k)$$

On note $Negl$ l'ensemble des fcts negl.

Remarque 1. On peut aussi prendre des grands O et des polynomes quelconques (réels).

Proposition 1.3.2. *$Negl$ est un $\mathbb{R}[X]$-module.(clair)*

mtn la bonne def:

Définition 1.3.3. Soit Π_S un schéma de signature numérique. On dit que le schéma est sécurisé asymptotiquement si pour tout algorithme probabiliste polynomial $\mathcal{A}$:

$$Pr(Exp_{\Pi,\mathcal{A}}(\lambda)) = neg(\lambda)$$

2 Chiffrement à clefs secrètes

Définition 2.0.1. Schéma de chiffrement à clef secrète (symétrique) : Triplet d'algo PPT

1. KeyGen(1^λ), le paramètre de complexité est λ
2. Enc(k, m)
3. Dec(k, c)

2.1 Sécurité sémantique

Définition 2.1.1. Soit $l \in \mathbb{N}$. On pose $\mathcal{M} = \{0,1\}^l$. Si pour tout PPT $\mathcal{A}$

$$Pr(\mathcal{A}_{m\leftarrow\mathcal{M}}(1^l, Enc(k,m)) = m^{(i)}) \leq \frac{1}{2} + negl(l)$$

2.2 Sécurité eav

On regarde le jeu $\mathbf{Priv}^{eav}_{\mathcal{A},\Pi}(\lambda)$:

1. $k \leftarrow \Pi.Keygen(1^lambda)$
2. $(m_0, m_1, \eta) \leftarrow \mathcal{A}_0$, le η définit l'état/la mémoire de $\mathcal{A}$. (pas obligé de la mettre)
3. $b \leftarrow \{0,1\}$ (choix aléatoire uniforme)
4. $b' \leftarrow \mathcal{A}_1(Enc_k(m_b), \eta)$
5. renvoie $b == b'$.

On peut juste écrire

1. $k \leftarrow \Pi.Keygen(1^lambda)$
2. $(m_0, m_1) \leftarrow \mathcal{A}$ (On suppose que c'est à $\mathcal{A}$ de choisir les messages qu'il veut distinguer)
3. $b \leftarrow \{0, 1\}$ (choix aléatoire uniforme)
4. $b' \leftarrow \mathcal{A}(Enc_k(m_b))$
5. renvoie $b == b'$.

On regarde aussi le jeu $\mathbf{Priv}^{eav2}_{\mathcal{A},\Pi}(\lambda)$:

1. $k \leftarrow \Pi.Keygen(1^lambda)$
2. $(m_0, m_1) \leftarrow \mathcal{A}$
3. $b^* \leftarrow \mathcal{A}(Enc_k(m_b))$
4. renvoie b^*.

Définition 2.2.1. Π est eav-securisé si pour tout PPT $\mathcal{A}$:

$$|Pr(PrivK^{eav}_{\mathcal{A},\Pi}(\lambda)) - 1/2| = negl(\lambda)$$

ou

$$|Pr(PrivK^{eav2}_{\mathcal{A},\Pi}(\lambda, 0)) - Pr(PrivK^{eav2}_{\mathcal{A},\Pi}(\lambda, 1))| = negl(\lambda)$$

2.3 exercice

Créer un modèle de sécurité pour un gestionnaire de mot de passe.

1. Init()
2. Ajoute(B, id, mdp)-¿(B', b')
3. Verify(B, id, mdp)

experience $Exp_{\Pi,\mathcal{A}}()$:

1. $\beta \leftarrow$Init()
2. $Q = \emptyset$

3. (id, mdp)$\leftarrow \mathcal{A}^{\mathcal{O}Ajoute_{inn},\mathcal{O}Acces}$(inn pour innocent)

4. renvoie $\Pi.Verify(\beta, id, mdp)\bigwedge$id$\notin Q$

$\mathcal{O}Acces()$ renvoie β et $\mathcal{O}Ajoute_{inn}(id)$:

1. mdp$\leftarrow \{0,1\}^*$

2. $(\beta', b') \leftarrow Ajoute(\beta, id, mdp)$

3. $\beta = \beta'$

4. renvoie b'

la sécurité ducoup c'est $Pr(Exp_{\Pi,\mathcal{A}}(1^n) = 1) < negl_n$

3 Générateurs pseudos aléatoires

Définition 3.0.1. $l : \mathbb{N} \to \mathbb{N}$ (polynome tq $l(n) > n$).
On dit que $G : \{0,1\}^* \to \{0,1\}^*$ un algo DPT est un géné pseudo aléatoires si $(G(\{0,1\}^n) \subset \{0,1\}^{l(n)})$:

1. $\forall$ PPT $\mathcal{A}$, $r \in \{0,1\}^n$ uniforme, $r' \in \{0,1\}^{l(n)}$ uniforme :

$$|Pr(\mathcal{A}(G(r))) - Pr(\mathcal{A}(r'))| \leq negl_n$$

Exos : etant donné $G(s)$ on déf

1. $G'(s) = G(\overline{s})$

2. $G'(s) = \overline{G(s)}$

3. $G'(s) = G(0^{|s|}||s)$

4. $G'(s) = G(s)||G(s+1)$

Maintenant les fonctions pseudo-aléatoires :

Définition 3.0.2. Soit $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ un DPT (meme longueur d'entrées sorties). F est pseudo aléatoire si $\forall$ D :

$$|Pr\left(D^{F_k(.)}() = 0\right) - Pr\left(D^{f()}() = 0\right)| = negl_n$$

(FPA pour fonction pseudo aléatoire)

On def $PrivK^{CPA-alea}_{\Pi_{FPA},\mathcal{A}}(\lambda, b)$ et $PrivK^{CPA}_{\Pi,\mathcal{A}}(\lambda, b)$ Ou le deuxieme c'est :

1. $k \leftarrow \{0,1\}^\lambda$
2. $(m_0, m_1) \leftarrow \mathcal{A}^{F_k()}$
3. $r^* \leftarrow \{0,1\}^n$
4. $b^* \leftarrow \mathcal{A}(r^*, m_b \oplus F_k(r^*))$, $(r^*, m_b \oplus F_k(r^*))$ est directement le cipher.

et le premier on remplace par une fonction aléatoire. (étant donné $r \in \{0,1\}^n$ on choisit $f(r) := r' \in \{0,1\}^n$, 2^{n2^n})

Au final on écrit

Définition 3.0.3. Pour tout distingueur avec oracle, le distingueur peut pas distinguer la FPA d'une fonction aléatoire.

Définition 3.0.4. De meme on suppose qu'on peut calculer l'inverse de la FPA. Alors F est une permutation pseudo aléatoire forte.

3.1 chiffrement par flot

Définition 3.1.1. Paire de DPT :

1. Init(s, IV), IV est connu et s est la sécurité. Renvoie un état η.
2. Nextbit(η) renvoie η' et un bit b.

Le chiffrement par flot est sécurisé? $JeuSC_{\mathcal{A},\Pi}(\lambda, b)$:

1. Choix de la seed $s \in \{0,1\}^\lambda$
2. $IV \leftarrow \mathcal{A}(1^\lambda)$
3. $\eta \leftarrow Init(s, IV)$
4. $b' \leftarrow \mathcal{A}^{\mathcal{O}}$

Ou l'oracle est soit : $\mathcal{O}()$: $(b', \eta') \leftarrow Nextbit(\eta)$, $\eta \leftarrow \eta'$, renvoie b'. Soit un bit aléatoire.

4 Fonctions universelles et application

On definit pour chaque $\lambda \in \mathbb{N}$: $h^{(\lambda)} \ : \ K_\lambda \times M_\lambda \to T_\lambda$ avec $(T_\lambda, +)$ un groupe. Soit $\epsilon \in [0,1]^{\mathbb{N}}$.

Définition 4.0.1. On dit que $h = (h^{(\lambda)})$ est une fonction universelle $\epsilon-$differentiable ssi $\forall \lambda \ \forall m_1, m_2, t \in M_\lambda^2 \times T_\lambda$:

$$Pr(h^{(\lambda)}(k, m_1) - h^{(\lambda)}(k, m_2) = t) \leq \epsilon(\lambda)$$

Ou la proba est prise sur $k \in K_\lambda$.

Soit $l \in \mathbb{N}$ et $(\mathbb{F}_\lambda)_\lambda$ une famille de corps de grande caractéristique. On def $h^{(\lambda)}(k \in \mathbb{F}_\lambda, (m_1, ..., m_{l'-1}) \in \mathbb{F}_\lambda^{l'-1}) = \sum_{i=1}^{l'-1} m_i k^{l'-i+1} + (l'-1)k$ avec $l' \leq l \leq car(\mathbb{F}_\lambda)$. Faut mq h est (l/λ)-differentiable (assez clair, d'ailleurs le $(l'-1)$ est la au cas ou on a deux messages $(0)^{l'}$, $(0)^{l''}$ et $l' \neq l''$).

Soit f une fpa et h une f-u $\epsilon-$diff avec $\epsilon = negl(\lambda)$. On pose

- $\Pi_{MACuniv}(1^\lambda).KeyGen()$:
  1. $k_h \leftarrow K_\lambda$
  2. $k_{fpa} \leftarrow \{0,1\}^\lambda$

  renvoie (k_h, k_{fpa}).
- MAC(k, m):
  - $r \leftarrow \{0,1\}^\lambda$
  - renvoie $(r, h(k_h, m) \oplus f(k_{fpa}, r))$
- $Verif_{k_h, k_{fpa}}(m, (t_1, t_2))$: renvoie $h_{k_h}(m) \oplus f_{k_{fpa}(t_1) == t_2}$.

Jeu : $Inforg_{\mathcal{A}, \Pi_{..}}(\lambda)$:

1. $k_f \leftarrow \{0,1\}^\lambda$
2. $k_h \leftarrow K_{lambda}$
3. $Q = \emptyset$
4. $(m^*, t^*) \leftarrow \mathcal{A}^{\mathcal{O}_{Mac}}$

5. renvoie $m^* \notin Q$, $t_2^* = F_k(t_1^*) + h_k(m^*)$

Avec $\mathcal{O}_{Mac}(m)$:

- $Q \leftarrow \{m\} \cup Q$
- renvoie (r,$F_{k_fpa}(r) + h_{k_h}(m)$)

  $Inforg_{\mathcal{A}}^{alea}(\lambda)$: (On peut mettre le choix de k_h APRES $m \notin Q$)

- $k_h \leftarrow K_{lambda}$
- $Q = \emptyset$
- $R = \emptyset$
- ...
- si $m^* \notin Q$
- si $(t, x_t, m_2) \in R$
- On renvoie $t_2^* = h_k(m^*) + x_t - h_k(m_2)$
- sinon on abandonne

avec $\mathcal{O}_{Mac}$:

- $Q = \{m\} \cup Q$
- $r \leftarrow \{0,1\}^\lambda$
- Si $r \in R$ on annule tout
- sinon $x \leftarrow \mathbb{F}_q$
- $R = R \cup \{(r, x, m)\}$
- renvoie (r, x).

5 Fonctions de Hashage

Définition 5.0.1. Etant donné $n \in \mathbb{N}$ on définit un couple:

1. $(Gen(1^\lambda),\ Hash : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^n)$

Qu'on appelle fonction de Hashage.

$Coll_H(\lambda)$:

$$\begin{aligned} &- s \leftarrow Gen(1^\lambda) \\ &- (m_0, m_1) \leftarrow \mathcal{A}(s) \\ &- renvoie(H^{(s)}(m) == H^{(s)}(m') \ \wedge \ m \neq m') \end{aligned}$$

Définition 5.0.2. On dit que H est resistante aux collisions si $\forall$ PPT $\mathcal{A}$:

$$P(Coll_{H,\mathcal{A}} = 1) = negl(\lambda)$$

5.1 Merkle-Daingard

Soit (Gen, h) une fonction de compression (Comme une fonction de hashage, mais qui prend des messages de taille $n' > n$ (la taille de la sortie)).

Définition 5.1.1. Soit $IV \in \{0,1\}^n$, on pose pour m_i de taille $n' - n$:

$$H^s(m_1 \dots m_L) = h^{(s)}(h^{(s)} \dots (h^{(s)}(h^{(s)}(IV||m_1)||m_2) \dots ||m_L)||L)$$

La sécurité venant de h se prouve par l'absurde, faut juste l'écrire.

6 Hash and MAC

Soit Π_{MAX} un MAC inforgeable et H une fonction de hashage résistante aux collisions.

Définition 6.0.1. On définit $\Pi_{Hash\&Mac}$ ou

$$\begin{aligned} &Gen : \ k \leftarrow \Pi_{MAC}.Gen, \ s \leftarrow H.Gen \\ &MAC_{k,s}(m) : \ \Pi_{MAC}.MAC(H.Hash(m)) \end{aligned}$$

Si H est vulnérable, une collision donne une forge. Si Π_{MAC} est vulnérable on trouve une collision.

Ca se formalise comme ça, $Coll_{\mathcal{A}}(s)$:

$$\begin{aligned}
&k \leftarrow Gen(\lambda) \\
&(m^*, t^*) \leftarrow \mathcal{A}^{\mathcal{O}_{Mac}}() \\
&si\ \exists \in Q\ tq\ H(m) = H(m^*)\ renvoie\ (m, m^*) \\
&sinon\ renvoie
\end{aligned}$$

avec $\mathcal{O}_{Mac}(m)$:

$$\begin{aligned}
&Q = Q \cup \{m\} \\
&h \leftarrow H^s(m) \\
&renvoie\ (Mac_k(h))
\end{aligned}$$

De l'autre côté $\mathcal{B}^{\mathcal{A},\mathcal{O}_{Mac'}}(1^\lambda)$:

$$\begin{aligned}
&s \leftarrow Gen(1^\lambda) \\
&(m^*, t^*) \leftarrow \mathcal{A}^{\mathcal{O}_{Mac'}} \\
&renvoie\ (H^s(m^*), t^*)
\end{aligned}$$

avec $\mathcal{O}_{Mac'}(m)$:

$$renvoie\ (\mathcal{O}_{Mac}(H^s(m)))$$

7 Preuve à divulgation nulle