Cryptographie

Signatures, certificats et PKIs

Carlos Aguilar

carlos.aguilar@enseeiht.fr

IRIT-IRT

Références

Livres électroniques

Cornell, https://www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf
Bristol, http://www.cs.umd.edu/~waa/414-F11/IntroToCrypto.pdf
Stanford, http://crypto.stanford.edu/~dabo/cryptobook/

Cours fortement inspiré de :

[1] Stanford, https://www.coursera.org/course/crypto
[2] Univ. of Virginia, http://www.udacity.com/view#Course/cs387/
[3] Univ. of Maryland, https://class.coursera.org/cryptography-006/

Plan

La signature électronique

Objectif initial

Assurer l'intégrité d'un message
Même principe que pour les MAC ... mais avec une tout autre dimension

Principe

Alice crée une bi-clé $(pub, priv)$ et publie *pub*
Des personnes obtiennent la clé d'Alice et s'assurent de son authenticité

Alice envoi des messages signés m, σ avec $\sigma = Sign(priv, m)$
$Verify(pub, m, \sigma)$ assure que σ a été produit avec la clé privée associée à *pub*

Mais il faut être sûr du lien *pub* $\leftrightarrow$Alice pour dire que c'est Alice qui a signé

Sécurité

Un attaquant voyant des couples message/signature ne peut pas produire un couple message/signature pour un nouveau message (sans connaître la clé privée)

Exemple d'utilisation [3]

Déploiement de patchs

Un éditeur avec une bi-clé (pub,priv) veut déployer un patch

- La clé publique pub est contenue dans le logiciel original
- Le (hash du) patch est signé par l'éditeur en utilisant priv et envoyé

Le programme de chaque utilisateur utilise pub pour vérifier l'authenticité du patch

Remarques

Tout patch Windows, Mac OSX, etc. utilise ce principe
De nos jours les éditeurs de logiciels s'enregistrent auprès des éditeurs d'OS...

Comparaison avec un MAC

Une même clé pour tous les utilisateurs $\rightarrow$ pas sûr
Une clé par utilisateur $\rightarrow$ cher

Et si l'éditeur est malveillant ? On a une preuve contre lui ! On parle de non-répudiation (infaisable avec un MAC), très important légalement

Schémas de signature : Définition

Algorithmes

Un cryptosystème à clé publique (PKC) est composé de trois algorithmes

- $KeyGen(1^k)$: alg. randomisé donnant en sortie une bi-clé $(pub, priv) \in K$
- $Sign(priv, m)$: alg. randomisé donnant en sortie un signature $\sigma \in S$ de m
- $Verify(pub, m, \sigma)$: alg. déterministe donnant en sortie true ou false

Consistance

$\forall k, \forall (pub, priv) \leftarrow KeyGen(1^k), \forall m \in M, Verify(pub, m, Sign(priv, m)) = true$

Sécurité : Existential Forgery

Comme pour les MAC + connaissance de *pub*
Peut obtenir des signatures de messages de son choix, puis doit générer un couple (m, σ) tq $Verify(pub, m, \sigma) = true$ pour un nouveau message m

Signature RSA-Raw

Pas bon du tout !

- Mauvais exemple trivial : signature de 1 ou de 0
- Mauvais exemple : forgery existentielle (couple antécédant/chiffré donne signature d'un nombre aléatoire)
- Mauvais exemple : homomorphisme (signature du produit de deux messages)

Pourquoi ?

Comment faire ?

Qu'est ce qui nous permet de passer d'un message structuré à un nombre au hasard ?

(Mais attention il faut respecter la sécurité (TF))

Signature RSA-Raw

Pas bon du tout !

- Mauvais exemple trivial : signature de 1 ou de 0
- Mauvais exemple : forgery existentielle (couple antécédant/chiffré donne signature d'un nombre aléatoire)
- Mauvais exemple : homomorphisme (signature du produit de deux messages)

Pourquoi ?

RSA permet pas de déchiffrer le chiffré d'un nombre au hasard or comme le chiffrement est une permutation ça équivaut à dire RSA ne permet pas de déchiffrer un nombre au hasard.

Comment faire ?

Qu'est ce qui nous permet de passer d'un message structuré à un nombre au hasard ?

(Mais attention il faut respecter la sécurité (TF))

RSA-FDH Crash course

Algorithmes

H fonction de hachage $\{0,1\}^* \rightarrow \{0,1\}^n$ pour $n = \log_2(N)$

- $KeyGen_{RSA-FDH}(1^k) = KeyGen_{PKC}(1^k)$
- $Sign_{RSA-FDH}(priv, m) = Dec_{PKC}(priv, H(m))$
- $Verify_{RSA-FDH}(pub, m, \sigma) = (H(m) == Enc_{PKC}(pub, \sigma))$

Consistance

Génération de la signature : $\sigma = Dec_{PKC}(priv, H(m)) = H(m)^d \bmod N$
Vérification de la signature :
$Enc_{PKC}(pub, \sigma) = H(m)^{d*e} \bmod N = H(m)^{1+const*\phi(N)} \bmod N = H(m) \bmod N$

Plan

Éditeurs d'OS et de logiciels (1/2)

Les éditeurs de logiciels

Pour chaque patch ils distribuent $(patch, \sigma_{patch})$ avec

$$\sigma_{patch} = Sign(priv_{\text{Éditeur}}, patch)$$

Clé publique dans le code mais comment la vérifier lors de la première installation ?
On ne peut pas connaître $pub_{\text{Éditeur}}$ pour chaque éditeur de logiciels ...

Racine de confiance

Les systèmes d'exploitation OS_Y ont la clé publique pub_{OS_Y} pré-enregistrée

Comment éviter que l'on doive connaître $pub_{\text{Éditeur}}$ pour chaque éditeur de logiciel ?

Éditeurs d'OS et de logiciels (2/2)

Enregistrement auprès de l'éditeur d'un OS

Chaque éditeur de logiciels EdLog qui veut distribuer des logiciels sous OS_Y

- s'enregistre auprès de l'éditeur de l'OS en founissant sa clé pub_{EdLog}
- l'éditeur OS génère : m="Clé de EdLog certifiée par OS_Y"$||pub_{\text{EdLog}}$
- l'éditeur OS envoie à EdLog $certif_{\text{EdLog}} = (m, \sigma_m)$ avec $\sigma_m = Sign(priv_{OS_Y}, m)$

Vérification d'un patch

Comment se passe la vérification d'un patch ?

Éditeurs d'OS et de logiciels (2/2)

Enregistrement auprès de l'éditeur d'un OS

Chaque éditeur de logiciels EdLog qui veut distribuer des logiciels sous OS_Y

- s'enregistre auprès de l'éditeur de l'OS en founissant sa clé pub_{EdLog}
- l'éditeur OS génère : m="Clé de EdLog certifiée par OS_Y"$||pub_{\mathrm{EdLog}}$
- l'éditeur OS envoie à EdLog $certif_{\mathrm{EdLog}} = (m, \sigma_m)$ avec $\sigma_m = Sign(priv_{OS_Y}, m)$

Vérification d'un patch

Comment se passe la vérification d'un patch ?

- on reçoit $(certif_{\mathrm{EdLog}}, patch, \sigma_{patch})$,
- on vérifie le certificat avec pub_{OS_Y}
- on vérifie la signature du patch avec la clé publique dans $certif_{\mathrm{EdLog}}$

$\Rightarrow$ On a bien un patch signé par un Éditeur certifié par OS_Y

Passage à l'échelle

Public Key Infrastructure : exemple hiérarchique

Considérons l'infrastructure suivante :

- $cert_{CNRS}$, $cert_{INRIA}$, $cert_{INPT}$, ... signés par la clé privée du MESR
- $cert_{IRIT}$, $cert_{LAAS}$, $cert_{LIP6}$, $cert_{LIX}$, $cert_{LIENS}$, ... signés par la clé privée du CNRS
- $cert_{CarlosAguilar}$, $cert_{EmmanuelChaput}$, ... signés par la clé privée de l'IRIT

Chaque $cert_X$ signé par Y sous la forme :
(m$||\sigma_m = Sign(priv_Y, m)$) avec m=("Clé de X signée par Y"$||pub_X$)

Vérification hiérarchique

pub_{MESR} dans tous les ordinateurs des fonctionnaires (racine de confiance)

Emmanuel envoie à Carlos ($m, \sigma_m, cert_{EmmanuelChaput}, cert_{IRIT}, cert_{CNRS}$). Carlos :

- utilise puv_{MESR} pour vérifier que $cert_{CNRS}$ est bien signé par le MESR
- utilise pub_{CNRS} pour vérifier que $cert_{IRIT}$ est bien signé par le CNRS
- utilise pub_{IRIT} pour vérifier que $cert_{EmmanuelChaput}$ est bien signé par l'IRIT
- utilise $pub_{EmmanuelChaput}$ pour vérifier que (m, σ_m) est bien signé par Emmanuel Chaput

L'infrastructure HTTPS de l'Internet

Autorités de Certification (CAs)

Chaque navigateur dispose d'une liste de certificats auxquels il fait confiance (car installés avec le navigateur)

- Ces autorités ont le droit d'associer n'importe quelle identité (URL) à n'importe quelle clé
- Il y a une multitude de CAs pour qu'il y ait de la concurrence et une bonne mise à l'échelle
- Quand on veut avoir un certificat reconnu par un navigateur on paye une CA reconnue par le navigateur pour qu'elle nous fasse un certificat pour notre clé+identité

Problèmes

Backdoors imposées par les gouvernements
Principe du maillon le plus faible
Contre : certificate pinning (spécifier un certificat ou une CA pour un site)

Plan

FIPS 186-4 : Digital Signature Standard (DSS)

Qu'est-ce ?

Standard universellement utilisé pour la signature électronique
Définit trois algorithmes possibles : RSA, DSA, ECDSA

Approches

- RSA
  - basé sur la factorisation ($k = 112 \Rightarrow \log_2 N = 2048$, etc.)
  - utilise un padding potentiellement randomisé, sinon principe proche de RSA-FDH
- DSA (Digital Signature Algorithm)
  - basé sur le log discret sur les corps finis ($k = 112 \Rightarrow \log_2 p = 2048$ et $\log_2 q = 224$, etc.)
  - suit une construction de type Fiat-Shamir (hors programme)
- ECDSA (Elliptic-Curve Digital Signature Algorithm)
  - basé sur le log discret sur les courbes elliptiques ($k = 112 \Rightarrow \log_2 p = 224$ sur courbe $P - 224$, etc.)
  - suit une construction de type Fiat-Shamir (hors programme)

Les certificats X.509 (1/2)

Hmmm

Format PEM (Privacy-enhanced Electronic Mail)
DER en base 64 (DER généralement en binaire illisible)

```
openssl s_client -connect www.laas.fr:443 |openssl x509  -outform pem
```


```
-----BEGIN CERTIFICATE-----
MIIEmzCCA4OgAwIBAgIQe0XYM05ernz0O9NhaPEwgzANBgkqhkiG9w0BAQsFADBk
MQswCQYDVQQGEwJOTDEWMBQGA1UECBMNTm9vcmQtSG9sbGFuZDESMBAGA1UEBxMJ
QW1zdGVyZGFtMQ8wDQYDVQQKEwZURVJFTkExGDAWBgNVBAMTD1RFUkVOQSBTU0wg
Q0EgMjAeFw0xNTA2MDIwMDAwMDBaFw0xODA2MDEyMzU5NTlaMDkxITAfBgNVBAsT
GERvbWFpbiBDb250cm9sIFZhbGlkYXRlZDEUMBIGA1UEAxMLd3d3LmxhYXMuZnIw
ggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCu9nkuSGKohk648wDzMBzI
A+DksCm6LgOZF9AbxVXJO9Z2q66CJMqwWN3Kuqn3zKUUt9v5Prw/23B/+uPw0qRF
5JStggDjWcq2i4/CgtSQU01umRiNg6mvqJS/pqryPAiw1fNJT/MiBRQ54oAGGg27
4ki2WVGByhTxvqf852NPi/8u5FT2CVeGbepr0ZVKfXWqqlnufhiypGEO7NXPL6dL
9pk9grTVaS1fW6Fyb4oQgNr1zway84bTEMP8G+dxPYDv7U6x6MrKRrcaKL+ngPF/
k8WvTkXfsiIIaykGDN0vT3Wb86FQuA1ecJfrpoTX60UF1hwcel4GX+r4DdgMQQt5
AgMBAAGjggFyMIIBbjAfBgNVHSMEGDAWgBRb0IocmjJb4LXdllQb4YYosP22vTAd
BgNVHQ4EFgQUzmAeAlP5wi/lkZRbxJ6lrlfigUUwDgYDVR0PAQH/BAQDAgWgMAwG
A1UdEwEB/wQCMAAwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMCIGA1Ud
IAQbMBkwDQYLKwYBBAGyMQECAh0wCAYGZ4EMAQIBMDoGA1UdHwQzMDEwL6AtoCuG
KWh0dHA6Ly9jcmwudXNlcnRydXN0LmNvbS9URVJFTkFTU0xDQTIuY3JsMGwGCCsG
AQUFBwEBBGAwXjA1BggrBgEFBQcwAoYpaHR0cDovL2NydC51c2VydHJ1c3QuY29t
L1RFUkVOQVNTTENBMi5jcnQwJQYIKwYBBQUHMAGGGWh0dHA6Ly9vY3NwLnVzZXJ0
cnVzdC5jb20wIQYDVR0RBBowGIILd3d3LmxhYXMuZnKCCSoubGFhcy5mcjANBgkq
hkiG9w0BAQsFAAOCAQEAonT2SV3igFAGhrxEG4/46uCO8XkdsxACHmp6yD2OxAYa
ZGAk75VjqBmyj2El+fMLHJPn/Q/NSGQXRsXBKqB+VtXIzHbdP0so3Z4fkEe/lj1Z
piVvf7oBb8O9n8ZR/bYokO7aK6POpvDK8R4uL1DqbSlxgkaE/avZfQzi/OJzGJbd
B6lTASpOQNOSwrwk2Yhv92YjohK/BlhaSnrPGMDS9OBzQ6BYPtGk+usyU12393nK
D8/joQu0Kgygw9Txbt40H2zyKzHTEF6ZgAE5eJ8dAjkAD5PM9uiQG46kUfcWfDtT
QtCEsKNzPltvdhOpRD/s9j/Yp6nX5zeaSt65TE71lw==
-----END CERTIFICATE-----
```

Les certificats X.509 (2/2)

Décodage du Abstract Syntax Notation 1

```
openssl s_client -connect www.laas.fr:443 |openssl x509 -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7b:45:d8:33:4e:5e:ae:7c:f4:3b:d3:61:68:f1:30:83
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=NL, ST=Noord-Holland, L=Amsterdam, O=TERENA, CN=TERENA SSL CA 2
        Validity
            Not Before: Jun  2 00:00:00 2015 GMT
            Not After : Jun  1 23:59:59 2018 GMT
        Subject: OU=Domain Control Validated, CN=www.laas.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ae:f6:79:2e:48:62:a8:86:4e:b8:f3:00:f3:30:
                    1c:c8:03:e0:e4:b0:29:ba:2e:03:99:17:d0:1b:c5:
                    55:c9:3b:d6:76:ab:ae:82:24:ca:b0:58:dd:ca:ba:
                    a9:f7:cc:a5:14:b7:db:f9:3e:bc:3f:db:70:7f:fa:
                    e3:f0:d2:a4:45:e4:94:ad:82:00:e3:59:ca:b6:8b:
                    8f:c2:82:d4:90:53:4d:6e:99:18:8d:83:a9:af:a8:
                    94:bf:a6:aa:f2:3c:08:b0:d5:f3:49:4f:f3:22:05:
                    14:39:e2:80:06:1a:0d:bb:e2:48:b6:59:51:81:ca:
                    14:f1:be:a7:fc:e7:63:4f:8b:ff:2e:e4:54:f6:09:
                    57:86:6d:ea:6b:d1:95:4a:7d:75:aa:aa:59:ee:7e:
                    18:b2:a4:61:0e:ec:d5:cf:2f:a7:4b:f6:99:3d:82:
                    b4:d5:69:2d:5f:5b:a1:72:6f:8a:10:80:da:f5:cf:
                    06:b2:f3:86:d3:10:c3:fc:1b:e7:71:3d:80:ef:ed:
                    4e:b1:e8:ca:ca:46:b7:1a:28:bf:a7:80:f1:7f:93:
                    c5:af:4e:45:df:b2:22:08:6b:29:06:0c:dd:2f:4f:
                    75:9b:f3:a1:50:b8:0d:5e:70:97:eb:a6:84:d7:eb:
                    45:05:d6:1c:1c:7a:5e:06:5f:ea:f8:0d:d8:0c:41:
                    0b:79
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         a2:74:f6:49:5d:e2:80:50:06:86:bc:44:1b:8f:f8:ea:e0:8e:
         f1:79:1d:b3:10:02:1e:6a:7a:c8:3d:8e:c4:06:1a:64:60:24:
         ef:95:63:a8:19:b2:8f:61:25:f9:f3:0b:1c:93:e7:fd:0f:cd:
         48:64:17:46:c5:c1:2a:a0:7e:56:d5:c8:cc:76:dd:3f:4b:28:
         dd:9e:1f:90:47:bf:96:3d:59:a6:25:6f:7f:ba:01:6f:c3:bd:
         9f:c6:51:fd:b6:28:90:ee:da:2b:a3:ce:a6:f0:ca:f1:1e:2e:
         2f:50:ea:6d:29:71:82:46:84:fd:ab:d9:7d:0c:e2:fc:e2:73:
         18:96:dd:07:a9:53:01:2a:4e:40:d3:92:c2:bc:24:d9:88:6f:
         f7:66:23:a2:12:bf:06:58:5a:4a:7a:cf:18:c0:d2:f4:e0:73:
         43:a0:58:3e:d1:a4:fa:eb:32:53:5d:b7:f7:79:ca:0f:cf:e3:
         a1:0b:b4:2a:0c:a0:c3:d4:f1:6e:de:34:1f:6c:f2:2b:31:d3:
         10:5e:99:80:01:39:78:9f:1d:02:39:00:0f:93:cc:f6:e8:90:
         1b:8e:a4:51:f7:16:7c:3b:53:42:d0:84:b0:a3:73:3e:5b:6f:
         76:13:a9:44:3f:ec:f6:3f:d8:a7:a9:d7:e7:37:9a:4a:de:b9:
         4c:4e:f5:97
```

La révocation (1/2)

Vol de certificat

Que se passe quand un certificat n'est plus valide ?
Partie du certificat malicieusement cachée par votre enseignant

```
X509v3 CRL Distribution Points:
  Full Name:
    URI:http://crl.usertrust.com/TERENASSLCA2.crl
  Authority Information Access:
    CA Issuers - URI:http://crt.usertrust.com/TERENASSLCA2.crt
    OCSP - URI:http://ocsp.usertrust.com
```

Principe des Listes de Révocation des Certificat (CRLs)
Pour valider un certificat il faut vérifier la signature, les champs (date, usage, etc.), ET vérifier qu'il n'a pas été révoqué :

- soit si on dispose d'une CRL non perimée
- soit en allant chercher la dernière CRL
- soit en utilisant OCSP (Online Certificate Status Protocol)

La révocation (2/2)

Notion du temps

Comment préserver les signatures d'avant vol d'une clé privée et ne pas accepter les signatures d'après (notification du) vol ?

→ Signatures avancées avec timestamp fourni par une TTP

Principe

Sur notre signature vient s'ajouter un timestamp et une signature par un serveur de temps qui va permettre de distinguer les deux cas

- signature générée (et timestamp ajouté) avant la révocation
- signature générée sans timestamp ou avec timestamp post-révocation

Si notre clé est compromise la signature reste valide tant que le serveur de temps ne voit pas sa clé privée compromise aussi

Fin !

Prochain cours

Application à SSL/TLS
Chiffrement authentifié ?
Pourquoi RSA-FDH ?
TD mise en place PKI ?