

Introduction à la Cryptographie

Buts et Moyens d'un attaquant

- 1 Principe fondamental de la cryptographie moderne
- 2 Pourquoi utiliser de la cryptographie ?
- 3 Modèles d'attaquants
- 4 Attaques par Canaux Cachés
- 5 Le modèle de Dolev-Yao
- 6 Que peut calculer un attaquant ?
- 7 Le paradigme de calcul quantique

Principe fondamental de la cryptographie moderne

Principe fondamental de la cryptographie moderne



*A cryptosystem should be secure even if **everything** about the system, except the secret key, **is public knowledge**.*
(August Kerckhoffs - 1883)

Les trois premiers items à retenir :

- ❶ **Sécurité calculatoire ou sémantique** : Le système doit être matériellement, sinon mathématiquement indéchiffrable ;
- ❷ **Transparence** : il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi ;
- ❸ **Portabilité** : La clef doit pouvoir en être communiquée et retenue sans le secours de notes écrites, et être changée ou modifiée au gré des correspondants.

Pourquoi utiliser de la cryptographie ?

Pourquoi utiliser de la cryptographie ?

- **Confidentialité** : L'information ne peut pas être lue par une personne non autorisée.

Pourquoi utiliser de la cryptographie ?

- **Confidentialité** : L'information ne peut pas être lue par une personne non autorisée.

Forward secrecy

Par exemple, en France, la confidentialité d'un document classifié doit être garantie pour une durée de 50 ans.

Pourquoi utiliser de la cryptographie ?

- **Confidentialité** : L'information ne peut pas être lue par une personne non autorisée.
- **Intégrité** : L'information ne peut être modifiée par une personne non autorisée.

Pourquoi utiliser de la cryptographie ?

- **Confidentialité** : L'information ne peut pas être lue par une personne non autorisée.
- **Intégrité** : L'information ne peut être modifiée par une personne non autorisée.
- **Authenticité** : L'information est attribuée à son auteur légitime.

Pourquoi utiliser de la cryptographie ?

- **Confidentialité** : L'information ne peut pas être lue par une personne non autorisée.
- **Intégrité** : L'information ne peut être modifiée par une personne non autorisée.
- **Authenticité** : L'information est attribuée à son auteur légitime.

Strong Authentication

L'authentification doit reposer au moins sur 2 éléments secrets connus seulement de l'entité à authentifier. Ça peut être quelque chose :

- ▶ qu'il **connait** (un mot de passe...);
- ▶ qu'il **possède** (une carte à puce...);
- ▶ qu'il **est** (biométrie...).

Pourquoi utiliser de la cryptographie ?

- **Confidentialité** : L'information ne peut pas être lue par une personne non autorisée.
- **Intégrité** : L'information ne peut être modifiée par une personne non autorisée.
- **Authenticité** : L'information est attribuée à son auteur légitime.
- **Non-répudiation** : L'information ne peut faire l'objet d'un déni de la part de son auteur.
 - ▶ **non-répudiation de l'origine** : L'émetteur ne peut pas nier avoir écrit le message et il peut prouver qu'il ne l'a pas fait si c'est effectivement le cas.
 - ▶ **non-répudiation de réception** : Le receveur ne peut pas nier avoir reçu le message et il peut prouver qu'il ne l'a pas reçu si c'est effectivement le cas.
 - ▶ **non-répudiation de transmission** : L'émetteur ne peut pas nier avoir envoyé le message et il peut prouver qu'il ne l'a pas fait si c'est effectivement le cas.

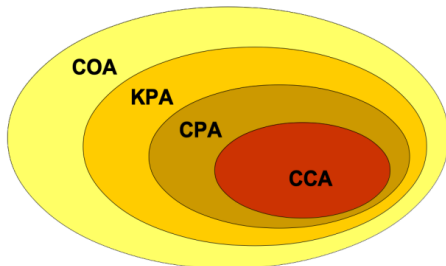
Modèles d'attaquants

Modèles d'attaquants

Selon le contexte, l'attaquant peut avoir accès à certaines informations pouvant l'aider à *casser* le crypto-système. Voici les modèles d'attaquants les plus connus :

- **Ciphertext-Only Attack (COA)** : L'attaquant connaît seulement un ou plusieurs chiffrés qu'il souhaite décrypter.
- **Known Plaintext Attack (KPA)** : L'attaquant connaît plusieurs couples clair/chiffré (qu'il n'a pas choisi lui-même).
- **Chosen Plaintext Attack (CPA)** : L'attaquant possède une machine chiffrente. Il peut donc chiffrer tous les messages qu'il souhaite.
- **Chosen Ciphertext Attack (CCA)** : L'attaquant possède une machine déchiffrente. Il peut donc choisir des chiffrés puis obtenir leurs messages clairs correspondants.

Modèles d'attaquants



Que peut-être le but de l'attaquant ?

- Trouver la clé secrète, la clé de déchiffrement.
- Plus modestement, décrypter (sans utiliser d'oracle) un message chiffré sans nécessairement découvrir la clé secrète.

Modèles d'attaquants

Quelles sont les pouvoirs de l'attaquant ? À quoi a-t-il accès ?

- **Cryptographie en boîte noire** : Les calculs sont effectués à distance. Selon le modèle, l'attaquant peut avoir accès aux entrées/sorties de la machine chiffrante ou déchiffrante mais pas à la machine elle-même (il ne peut pas observer la machine durant ses exécutions).
- **Cryptographie en boîte blanche** : L'attaquant connaît chaque étape de l'exécution de l'algorithme de chiffrement/déchiffrement (reverse engineering, accès à un debugger...).
- **Cryptographie en boîte grise** : Attaques par canaux cachés ou *Side channel attack*. L'attaquant a accès à des informations partielles qui ont fuité lors de l'exécution l'algorithme de chiffrement/déchiffrement.

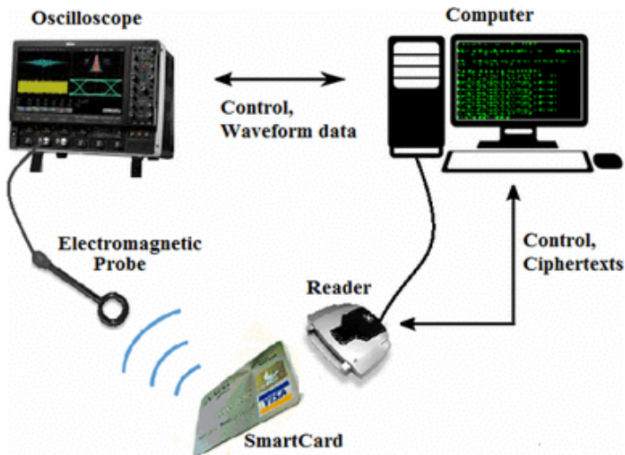
Attaques par Canaux Cachés

Attaques par Canaux Cachés

Initié par Paul Kocker dans les années 1990.

- **Time attack** : L'attaquant peut mesurer le temps d'exécution de l'algorithme de chiffrement/déchiffrement.
- **Power attack** : L'attaquant peut mesurer la consommation électrique durant l'exécution de l'algorithme de chiffrement/déchiffrement.
- **Electromagnetic attack** : L'attaquant peut mesurer le rayonnement électromagnétique durant l'exécution de l'algorithme de chiffrement/déchiffrement.
- **Micro-Ultrasound attack** : L'attaquant peut mesurer les micro-ultrasons durant l'exécution de l'algorithme de chiffrement/déchiffrement.

Attaques par Canaux Cachés



Attaques par Canaux Cachés

Exercice : time attack

Supposons une SmartCard utilisée pour un système d'authentification. L'utilisateur envoie un code PIN à la SmartCard qui retourne Vrai si le code PIN est le même que celui qui est sauvegardé dans la carte et Faux sinon.

On suppose que l'on n'a pas accès au code Python de la carte. Pour simuler l'utilisation de la carte, la seule chose que l'on peut faire est appeler la fonction `Authentication()`.

- Proposer une attaque pour retrouver la taille du code PIN puis le code PIN lui-même.
- Proposer ensuite une contre-mesure.

Attaques par Canaux Cachés

Exercice : time attack / Code de la SmartCard

```
# global variable
savedPIN="abcdefgh"

def Authentication(PIN):
    if len(PIN) != len(savedPIN):
        return False
    else:
        for i in range(len(PIN)):
            sleep(0.001) # to better see the attack
            if PIN[i] != savedPIN[i]:
                return False
        sleep(0.001) # to better see the attack
        return True
```

Attaques par Canaux Cachés

Corrigé : time attack / Mesurer la taille du code PIN

```
timestamp_max = [0,0]
for size in range(20):
    PIN_test = "a"*size
    time_start = time()
    Authentication(PIN_test)
    timestamp = time() - time_start
    if timestamp > timestamp_max[1] :
        timestamp_max[0] = size
        timestamp_max[1] = timestamp
size = timestamp_max[0]
print("The_size_is", size)
```

Attaques par Canaux Cachés

Corrigé : time attack / Retrouver le code PIN

```
PIN = ""
for i in range(size):
    # the caractere with the greatest timestamp
    timestamp_max = ['a', 0]
    for c in [chr(j) for j in range(ord('a'), ord('z')+1)]:
        PIN_test = PIN + c + ('a'*(size - len(PIN) - 1))

        time_start = time()
        Authentication(PIN_test)
        timestamp = time() - time_start

        if timestamp > timestamp_max[1] :
            timestamp_max[0] = c
            timestamp_max[1] = timestamp
    PIN += timestamp_max[0]
print(PIN)
```

Attaques par Canaux Cachés

Corrigé : time attack / Contre-mesure 1 : temps d'exécution décorrélié du secret

```
def Authentication_randTime(PIN):  
    if len(PIN) != len(savedPIN):  
        return False  
    else:  
        sleep(uniform(0.0, 0.01)) # add a random time  
        for i in range(len(PIN)):  
            sleep(0.001) # to better see the attack  
            if PIN[i] != savedPIN[i]:  
                return False  
        sleep(0.001) # to better see the attack  
    return True
```

Attaques par Canaux Cachés

Corrigé : time attack / Contre-mesure 2 : temps d'exécution constant

```
def Authentication_constTime(PIN):  
    flag = True  
    if len(PIN) != len(savedPIN):  
        flag = False  
    else :  
        for i in range(len(PIN)):  
            sleep(0.001) # to better see the attack  
            if PIN[i] != savedPIN[i]:  
                flag = False  
            sleep(0.001) # to better see the attack  
    return flag
```


Le modèle de Dolev-Yao

Le modèle de Dolev-Yao

C'est un modèle souvent considéré en réseau. Notamment du fait que n'importe qui dans le réseau peut faire de l'ARP¹ spoofing (attaque de l'homme du milieu). Dans ce modèle, on suppose que l'attaquant :

- peut obtenir tous les messages circulant sur le réseau ;
- peut initier une conversation avec n'importe quel membre du réseau ;
- peut envoyer un message à n'importe quel membre du réseau en se faisant passer pour n'importe quel membre du réseau ;
- **ne peut pas** deviner un nombre entier qui a été tiré aléatoirement ;
- **ne peut pas** déterminer la clé privée associée à une clé publique.

Que peut calculer un attaquant ?

Que peut calculer un attaquant ?

Deux notions de “sécurité” :

- ① Le designer veut atteindre une **sécurité inconditionnelle** : Il peut prouver que son cryptosystème est sûr sans préjuger de la puissance de calcul d'un éventuel attaquant qui peut même être infinie !
→ En particulier, si un couple clair/chiffré ne donne aucune information sur la clé, alors nous pouvons dire que le cryptosystème est **parfaitement sécurisé**.
- ② La **sécurité calculatoire** est basée sur l'impossibilité de décrypter un message ou de retrouver la clé secrète en un temps raisonnable, étant donnée la puissance de calcul d'un éventuel attaquant. Cette notion de sécurité dépend de l'état-de-l'art à un moment donné.
→ Pour atteindre une sécurité calculatoire, nous avons besoin de **preuve de sécurité** ou plus exactement de **réduction de sécurité**. Cela consiste à réduire le fait de “casser le cryptosystème” à “résoudre un problème mathématiques difficile”.

Que peut calculer un attaquant ?

Qu'est-ce qu'un problème difficile ?

Étant donné un problème prenant en entrée une chaîne binaire de longueur n , la complexité temporelle pour le résoudre est :

- **linéaire** : nécessite $an + b = O(n)$ opérations.
- **polynomiale** : nécessite $a_0 + a_1n + a_2n^2 + \dots + a_dn^d = O(n^d)$ opérations avec d fixé.
- **exponentielle** : nécessite $O(2^{\alpha n})$ opérations avec α fixé.

Que peut calculer un attaquant ?

Qu'est-ce qu'un problème difficile ?

- Un problème (de décision) pour lequel on peut calculer une solution en un temps polynomial est un problème de la classe P (Polynomial).
- Un problème (de décision) pour lequel on peut "vérifier" une solution en un temps polynomial est un problème de la classe NP (Non-deterministe polynomial).

→ **Un problème à \$1M : $P = NP$?**

Que peut calculer un attaquant ?

Qu'est-ce qu'un problème difficile ?

- Un problème (de décision) pour lequel on peut calculer une solution en un temps polynomial est un problème de la classe P (Polynomial).
- Un problème (de décision) pour lequel on peut "vérifier" une solution en un temps polynomial est un problème de la classe NP (Non-deterministe polynomial).

→ **Un problème à \$1M** : $P = NP$? Probablement pas ...

Que peut calculer un attaquant ?

Qu'est-ce qu'un problème difficile ?

- Un problème (de décision) pour lequel on peut calculer une solution en un temps polynomial est un problème de la classe P (Polynomial).
- Un problème (de décision) pour lequel on peut "vérifier" une solution en un temps polynomial est un problème de la classe NP (Non-deterministe polynomial).

→ **Un problème à \$1M** : $P = NP$? Probablement pas ...

Les problèmes **NP -complets** forment une sous-classe de NP . Un problème est NP -complet si tous les problèmes NP sont au moins aussi durs que lui. En d'autres mots, un problème A est NP -complet si pour tout problème NP B , il y a une réduction polynomiale de B à A .

Que peut calculer un attaquant ?

Qu'est-ce qu'un problème difficile ?

- Un problème (de décision) pour lequel on peut calculer une solution en un temps polynomial est un problème de la classe P (Polynomial).
- Un problème (de décision) pour lequel on peut "vérifier" une solution en un temps polynomial est un problème de la classe NP (Non-deterministe polynomial).

→ **Un problème à \$1M** : $P = NP$? Probablement pas ...

Les problèmes **NP -complets** forment une sous-classe de NP . Un problème est NP -complet si tous les problèmes NP sont au moins aussi durs que lui. En d'autres mots, un problème A est NP -complet si pour tout problème NP B , il y a une réduction polynomiale de B à A .

⇒ Pour prouver $P = NP$, il suffit de choisir un problème NP -complet et de montrer qu'il est P .

Que peut calculer un attaquant ?

Qu'est-ce qu'un problème difficile ?

- Un problème (de décision) pour lequel on peut calculer une solution en un temps polynomial est un problème de la classe P (Polynomial).
- Un problème (de décision) pour lequel on peut "vérifier" une solution en un temps polynomial est un problème de la classe NP (Non-deterministe polynomial).

→ **Un problème à \$1M** : $P = NP$? Probablement pas ...

Les problèmes **NP -complets** forment une sous-classe de NP . Un problème est NP -complet si tous les problèmes NP sont au moins aussi durs que lui. En d'autres mots, un problème A est NP -complet si pour tout problème NP B , il y a une réduction polynomiale de B à A .

⇒ Pour prouver $P = NP$, il suffit de choisir un problème NP -complet et de montrer qu'il est P .

Pour conclure, on veut que nos cryptosystèmes se réduisent à des problèmes NP -complets.



Que peut calculer un attaquant ?

Qu'est-ce qu'un problème difficile ?

Contrairement à la croyance populaire, le problème de **factoriser un entier**, du **logarithme discret sur un anneau d'entiers** ou du **logarithme discret sur une courbe elliptique** (problèmes sur lesquels repose la cryptographie asymétrique aujourd'hui) sont *NP* mais **ne sont pas NP-complets** !

- La factorisation est plus complexe à mettre en œuvre :
 - ▶ Combien font 23×53 ?
 - ▶ Factorisez 1363.
- La multiplication de deux entiers est de complexité d'ordre de $O(n^2)$
- La meilleure méthode de factorisation d'un entier est de complexité d'ordre de $O(\exp(4n^{1/3}))$

Que peut calculer un attaquant ?

Définition (bits de sécurité)

Lorsque que l'on dit qu'un cryptosystème a x bits de sécurité, cela signifie qu'un attaquant à besoin d'effectuer au moins $O(2^x)$ opérations pour le casser.

- $O(2^{20})$: nombre d'opérations simples que peut effectuer votre ordinateur personnel en un temps raisonnable (entre une seconde et un mois).
- $O(2^{50})$: nombre de mots de passe que des super-calculateurs (grosses sociétés, organismes militaires...) peuvent casser en un temps raisonnable ($\simeq 1$ mois).
- $\geq 2^{80}$: limite que l'on considère sûre aujourd'hui.
- $\geq 2^{128}$ ou $\geq 2^{256}$: ce que les organismes de normalisation (NIST...) demandent.

Que peut calculer un attaquant ?

Théorème (bits de sécurité et taille de clé)

Si la clé secrète que l'on veut retrouver s'écrit sur n bits, alors le cryptosystème a **au plus** n bits de sécurité.

Preuve : Attaque Brute force

Attention, le nombre de bits de sécurité peut être plus petit que la taille de la clé.

Le paradigme de calcul quantique

Le paradigme de calcul quantique

Le modèle d'attaquant quantique





Un attaquant ayant accès à un ordinateur quantique définit un nouveau modèle d'attaquant.

Les ordinateurs quantiques nous plongent dans un nouveau monde où :

- La *NP*-complétude n'a plus de sens !
- Le problème de factoriser un entier et le problème du log discret sur un anneau d'entier ou sur une courbe elliptique peuvent être résolus en temps polynomial avec l'**algorithme de Shor** (1994).
- Chercher un élément particulier dans un ensemble non-structuré de 2^n éléments est un problème qui ne demande que $O(2^{n/2}) = O(\sqrt{2^n})$ opérations grâce à l'**algorithme de Grover** (1996).

Le paradigme de calcul quantique

Un ordinateur classique, comment ça marche ?

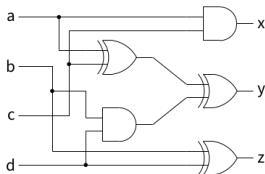
Porte logique	Symbole électronique	Table de vérité															
NO		<table><tr><th>in</th><th>out</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	in	out	0	1	1	0									
in	out																
0	1																
1	0																
AND		<table><tr><th>in 1</th><th>in 2</th><th>out</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	in 1	in 2	out	0	0	0	0	1	0	1	0	0	1	1	1
in 1	in 2	out															
0	0	0															
0	1	0															
1	0	0															
1	1	1															
OR		<table><tr><th>in 1</th><th>in 2</th><th>out</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	in 1	in 2	out	0	0	0	0	1	1	1	0	1	1	1	1
in 1	in 2	out															
0	0	0															
0	1	1															
1	0	1															
1	1	1															
XOR		<table><tr><th>in 1</th><th>in 2</th><th>out</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	in 1	in 2	out	0	0	0	0	1	1	1	0	1	1	1	0
in 1	in 2	out															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

Légende :

1 : le courant passe

0 : le courant ne passe pas

Exemple d'un programme :



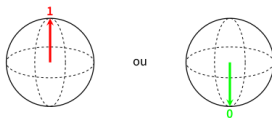
Effectue l'addition en base 2 :

$$\begin{array}{r}
 \\
 + \\
 \hline
 =
 \end{array}
 \begin{array}{cc}
 a & b \\
 c & d \\
 y & z
 \end{array}$$

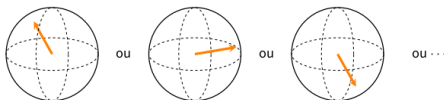
Le paradigme de calcul quantique

Bit vs Qubit (sphère de Bloch)

Un **bit** est **1** ou **0** :



Un **qubit** est à la fois **1** et **0** :

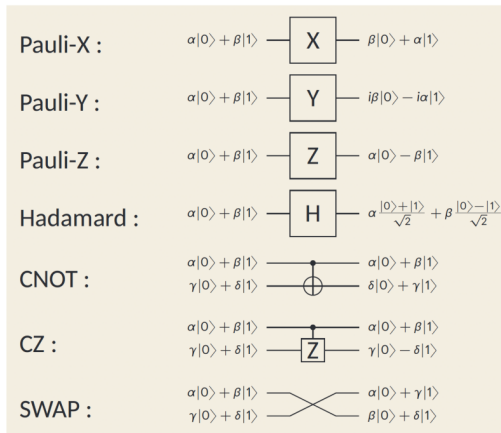


Il n'y a que lors de la **mesure** que l'on peut déterminer si le qubit est :

- dans l'hémisphère nord : on considère alors la valeur **1**
- dans l'hémisphère sud : on considère alors la valeur **0**

Le paradigme de calcul quantique

On considère de nouvelles **portes** dites **quantiques** :



... et donc on a une algorithmique différente.

Le paradigme de calcul quantique

Des algorithmes quantiques célèbres :

- **Grover 1996** : trouver un élément particulier dans un ensemble de N éléments non structurés en \sqrt{N} opérations.
- **Shor 1994** : factoriser un entier naturel N en $\log(N)^3$ opérations.
L'algorithme de Shor peut également être utilisé pour résoudre le problème du Logarithme Discret sur \mathbb{Z}_n ou sur les Courbes Elliptiques.

Le paradigme de calcul quantique

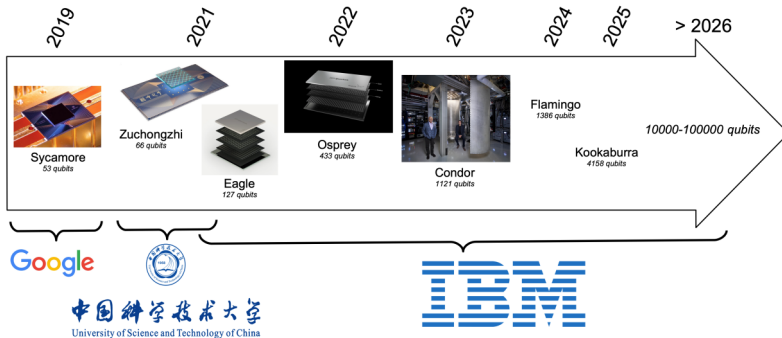
Le problème de la décohérence

Plus le nombre de qubits est grand, plus il est difficile de les maintenir dans un état d'intrication et de superposition quantique suffisamment longtemps.

- protéger l'environnement de calcul : refroidir au **zéro absolu**, ...
- utiliser des **codes correcteurs quantiques** : les *codes de surface* nécessitent un grand nombre de qubits physiques par qubit logique.

Le paradigme de calcul quantique

Où en est-on ?

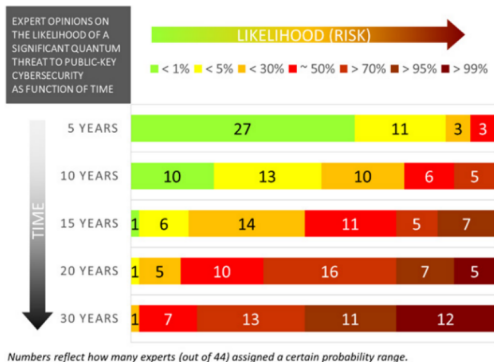


qubits logiques / qubits physiques

Sur la figure ci-dessus, on dénombre des **qubits physiques**. Il est nécessaire de combiner de nombreux qubits physiques (sujets aux erreurs) pour obtenir un **qubit logique** (sans erreur). On parle de code correcteurs quantiques.

Le paradigme de calcul quantique

Où en est-on ?



forward secrecy

Même si le nombre de qubits logiques est insuffisant aujourd'hui pour casser RSA, on doit se protéger dès maintenant pour garantir le **forward secrecy**.