

# Blockchain - Web3

Mathieu Bour

Mewo Informatique



# Le (sublime) formateur (c'est moi !)

- Ingénieur spécialisé en Blockchain/Web3
- Investisseur dans les cryptomonnaies depuis 2013
- Auditeur de smart-contracts depuis 2021
- (accessoirement) diplômé des Mines de Saint-Étienne en 2020
- Levé \$3M avec DeepSquare sur la blockchain Avalanche
- Actuellement chez Pooky, un jeu de prédiction de match de foot



Mathieu Bour

Tél : 06.95.39.72.53

Mail (Mewo) : [mathieu.bour@mewo-campus.fr](mailto:mathieu.bour@mewo-campus.fr)

Mail (pro) : [mathieu@bour.tech](mailto:mathieu@bour.tech)



- DYOR = « Do Your Own Research » : bien que ce cours soit à jour en mai 2023, je peux m'être trompé. La blockchain n'autorisant pas l'erreur, prenez le temps de faire vos propres recherches.
- La blockchain peut permettre de gagner beaucoup d'argent, mais la très grande majorité des investisseurs perdent leur mise.DYOR.
- En tant que pro-décentralisation, certaines slides peuvent ne pas être objectives, voire tomber dans la poilitique.DYOR.
- Bien que nous évoquerons l'ensemble des types de blockchains, nous utiliserons uniquement les blockchains publiques et décentralisées dans ce cours.

# Sommaire général (WIP)

- 1 Cryptographie
- 2 Introduction à la blockchain
- 3 Blockchain Ethereum et smart contracts

# Cryptographie

# Sommaire

## 1 Cryptographie

- Hachage
- Chiffrement symétrique
- Chiffrement asymétrique
- Signature numérique

## 2 Introduction à la blockchain

## 3 Blockchain Ethereum et smart contracts

# Objectifs de ce module

Comprendre les opérations de base de la cryptographie

- 1 Hachage
- 2 Chiffrement symétrique/asymétrique
- 3 Signature digitale

# Notations

*Je suis désolé, il faut faire un tout petit peu de maths...*

Dans la suite, je vais noter :

- $\mathbb{B} = \{0, 1\}^{\mathbb{N}}$  l'ensemble des mots binaires
- $\mathbb{B}_{n \in \mathbb{N}} = \{0, 1\}^n$  l'ensemble des mots binaires de taille  $n$

Exemples :

- $B_3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
- 010010 est un mot binaire de 6 bits, il est donc membre de  $\mathbb{B}_6$




# Qu'est-ce que la cryptographie ?

TL ;DR = utiliser les mathématiques au service de la sécurité de l'information

Exemples historiques :

- Chiffrement de César : décalage de lettre de 1 à 25
- Chiffrement de Vigenère : substitutions de lettres à partir d'une clé secrète
- Chiffrement affine : substitution de lettre à l'aide d'une équation affine
- Enigma (seconde guerre mondiale) : machine de chiffrement allemande



Figure – Machine Enigma 

# Somme de contrôle : définition

## Définition : somme de contrôle

Une somme de contrôle est une petite quantité de données additionnelle qui est calculée à partir d'un ensemble plus large de données. Elle est utilisée pour vérifier l'intégrité des données et détecter les erreurs ou les altérations éventuelles.

Exemples :

- Numéro de sécurité sociale
- Numéro de carte bancaire
- IBAN
- Mémoire ECC (Error Correcting Code)

# Somme de contrôle : exemple du numéro de sécurité sociale

Les deux derniers chiffres du numéro de sécurité sociale ne contiennent aucune information mais ils sont utilisés comme somme contrôle, pour limiter les risques de faute d'erreur.

La formule permettant de calculer la clé est la suivante :

$$\text{clé} = 97 - \text{NIR} \bmod 97$$

$$\underbrace{2690549588157}_{\text{numéro NIR}} \underbrace{80}_{\text{clé}}$$

```
1 >>> 97 - 2690549588157 % 97
2 80
```

# Fonction de hachage : définition

Comment appliquer cette logique à de l'information binaire? On cherche une somme de contrôle universelle capable de fonctionner sur tout  $\mathbb{B}$ .

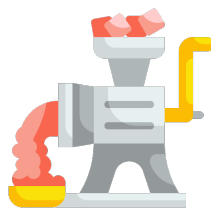
⇒ on les appelle fonction de hachage

## Définition : fonction de hachage

Une fonction de hachage permet de générer un « hash » de n'importe quel mot binaire.

## Définition : hash

Un hash est un mot binaire de taille fixe, dont la taille est spécifique à la fonction de hachage utilisé.



# Fonction de hachage : exemples

```
1 >>> import hashlib
2 >>> hashlib.md5(b"Mathieu").hexdigest()
3 '206299fa740a4327a61b67b6be5c8373'
```

Figure – Calcul d'un MD5 en ligne de commande en Python

```
1 >>> import hashlib
2 >>> hashlib.sha256(b"Mathieu").hexdigest()
3 'f5e088d29801ebb822251d7751bc4b8ff28c50132d8b0a95614b5f048a1d01b6'
```

Figure – Calcul d'un SHA-256 en ligne de commande en Python

# Caractéristiques d'une fonction de hachage

## Uniformité

Une bonne fonction de hachage doit répartir uniformément les valeurs d'entrée sur l'ensemble des valeurs de hachage possibles. Cela signifie que des entrées différentes doivent avoir une probabilité égale de générer des valeurs de hachage différentes.

## Déterminisme

Pour une même valeur d'entrée, la fonction de hachage doit toujours générer la même valeur de hachage. Cela permet d'obtenir des résultats cohérents et reproductibles.

# Caractéristiques d'une fonction de hachage

## Résistance aux collisions

Une fonction de hachage  $h$  de taille  $n$  entraîne obligatoirement des collisions car la taille de  $\mathbb{B}$  est infinie alors que  $\mathbb{B}_n$  n'est « que » de  $2^n$ . Une collision existe quand deux mots binaires  $a$  et  $b$  engendrent le même hash, c'est-à-dire :

$$h(a) = h(b)$$

⇒ une « bonne » fonction de hachage ne possède pas de hash connu.

*Note : Les algorithmes md4, md5 et sha1 ne sont à jour plus considérés comme sûrs.*

# Caractéristiques d'une fonction de hachage

## Sensibilité aux changements

Une petite modification dans l'entrée doit entraîner un changement significatif dans la valeur de hachage. Cela garantit que des entrées similaires génèrent des valeurs de hachage différentes.

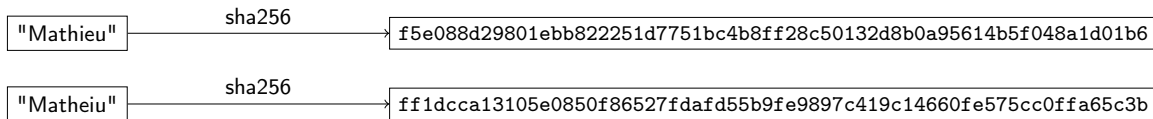


Figure – Sensibilité aux changements de la fonction SHA-256



# Caractéristiques d'une fonction de hachage

## Efficacité

Une bonne fonction de hachage doit être rapide à calculer pour des données de toutes tailles. Les performances de la fonction de hachage sont essentielles, car elle est souvent utilisée dans des applications nécessitant des opérations rapides sur de grandes quantités de données.

Ces performances sont tellement cruciales que certains algorithmes de hachage sont directement implémentés en tant qu'instructions dans les processeurs. Par exemple, les algorithmes SHA-1 et SHA-256 sont **matériellement** présents dans tous les processeurs Intel depuis 2013.

# Caractéristiques d'une fonction de hachage

## Le plus important : résistance aux attaques

Une fonction de hachage sécurisée doit être résistante à différentes attaques, telles que :

- les attaques de collisions
- les attaques de préimage
- les attaques par force brute

Elle doit être suffisamment robuste pour empêcher un adversaire de trouver des **collisions intentionnelles** ou de retrouver l'**entrée d'origine à partir de la valeur de hachage**.

# Fonction de hachage : résumé

- Une fonction de hachage prend n'importe quel mot binaire entrée et retourne un mot binaire de taille fixe
- Une bonne fonction de hachage possède les caractéristiques suivantes :
  - Uniformité
  - Déterminisme (= pas de hasard)
  - Résistance aux collisions
  - Sensibilité aux changements
  - Efficacité
  - **Résistance aux attaque bruteforce / reverse**

# Fonction de hachage : exercices

## Exercice 1 : se familiariser avec les fonctions de hachage

En CLI, calculez le hash de votre prénom / animal / n'importe quoi. Vous pouvez utiliser des bibliothèques ou des outils en ligne pour effectuer ce calcul. Comparez ensuite la valeur de hachage obtenue avec celle d'autres chaînes de caractères. Observez comment une légère modification de la chaîne d'entrée entraîne un changement significatif dans la valeur de hachage.

## Exercice 2 : calcul du hash d'un fichier

Calculez le hash du whitepaper du bitcoin, accessible sur <https://bitcoin.org/bitcoin.pdf>. Comparez sa valeur avec les autres.

# Chiffrement : définition

## Définition : chiffrement

Le chiffrement est une technique permettant à deux parties d'échanger de manière sécurisée. Il existe deux grands types de chiffrements, dits symétrique et asymétrique.

Les principales caractéristiques du chiffrement sont :

- Confidentialité : protéger contre l'accès non autorisé, seules les personnes ayant la clé puissent déchiffrer et lire le message.
- Intégrité : détecter toute modification ou altération des données chiffrées. Si les données chiffrées sont altérées, le déchiffrement donnera un résultat incorrect ou une erreur.
- Efficacité : traiter rapidement les données, en particulier lorsqu'il s'agit de volumes importants.
- Sécurité : être résistants aux attaques cryptographiques, telles que les attaques par force brute, les attaques de collision, différentielles. . .

# Modèle de canal de communication

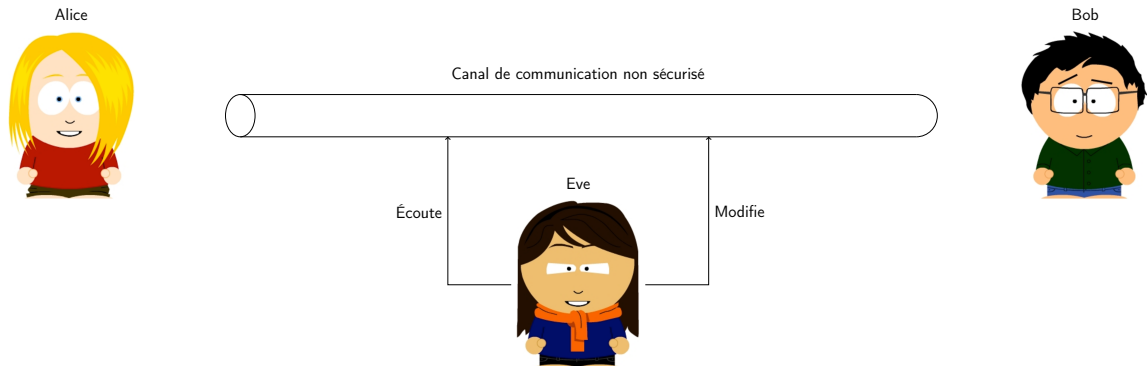


Figure – Modèle de canal de communication non sécurisé

# Chiffrement symétrique

## Définition : chiffrement symétrique

Le chiffrement symétrique est une technique de chiffrement où **une seule et même clé** est utilisée à la fois pour le **chiffrement** et le **déchiffrement** des données. Cela signifie que l'émetteur et le destinataire du message doivent partager la même clé secrète pour pouvoir communiquer de manière sécurisée.

Exemples d'algorithmes de chiffrement symétriques :

- AES
- DES (obsolète) et triple DES

# Chiffrement symétrique : modèle

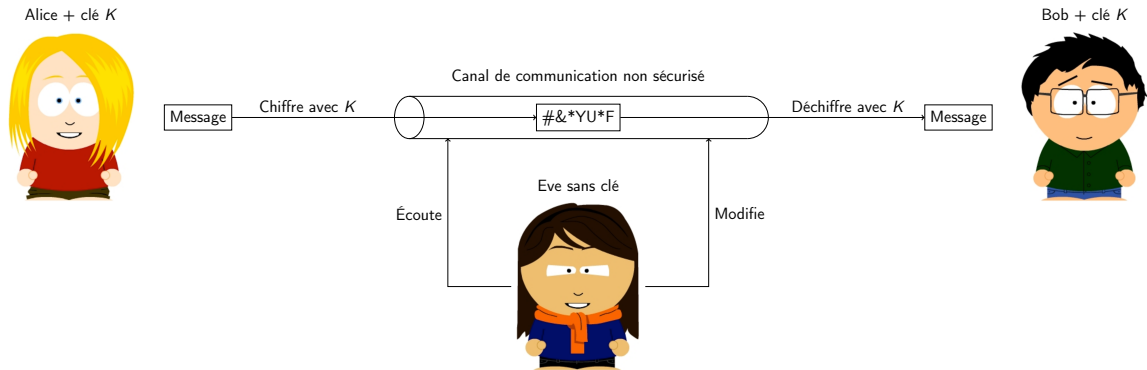


Figure – Chiffrement symétrique avec une clé  $K$



# Chiffrement symétrique : exercice

## Exercice : échange d'information sur canal public

- ➊ Aller sur <https://www.devglan.com/online-tools/aes-encryption-decryption>
- ➋ Choisir une clé de 32 caractères hexadécimaux (par ex : 770A8A65DA156D24EE2A093277530142).
- ➌ Partager la clé avec un ami.
- ➍ Chiffrer un message et le publier sur un canal public.
- ➎ Vérifier que l'ami est capable de déchiffrer le message et personne d'autre.

# Chiffrement symétrique : résumé

- Le chiffrement symétrique permet d'échanger des messages secrets.
- Une seule clé pour chiffrer et déchiffrer  $\Rightarrow$  l'émetteur et le destinataire doivent connaître la clé.
- AES est l'algorithme le plus utilisé.

Problème : comment partager la clé de manière sécurisée ?

# Chiffrement asymétrique

## Définition : chiffrement asymétrique

Le chiffrement asymétrique, également connu sous le nom de cryptographie à clé publique, est une technique de chiffrement qui utilise une paire de clés distinctes pour le processus de chiffrement et de déchiffrement des données.

Cette paire de clés est composée d'une clé publique et d'une clé privée. Elles sont liées mathématiquement. On peut alors chiffrer un message avec la clé publique et déchiffrer le message avec la clé privée.

Cela permet de recevoir des messages de manière sécurisée sans échange de clé au préalable.

- 1 Alice publie sa clé publique (par exemple sur son profil GitHub).
- 2 Bob va la contacter et lui envoie un message chiffré avec la clé publique d'Alice
- 3 Alice déchiffre le message avec sa clé privée.

# Chiffrement asymétrique : exemple

On considère un algorithme de chiffrement asymétrique générique :

```
1  def cipher(data: str, key: str) -> str:
2  def decipher(data: str, key: str) -> str:
3
4  message = "Hello world"
5  private_key, public_key = generateKeys()
6
7  result = cipher(message, public_key) # "U*#U*#QQ#"
8  clear = decipher(result, private_key) # "Hello world"
```

# Chiffrement asymétrique : communication sécurisée

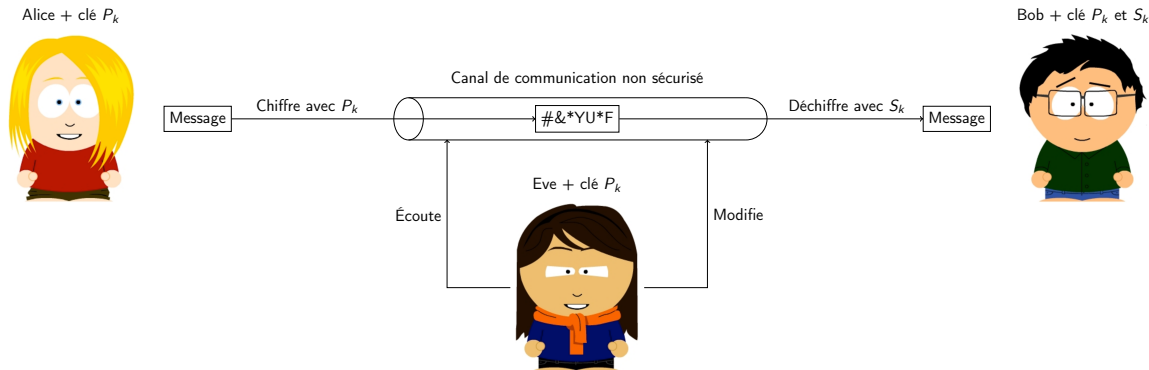


Figure – Chiffrement asymétrique avec une clé ( $P_k, S_k$ )

# Combinaison asymétrique/symétrique

L'utilisation des deux techniques de chiffrement permet de communiquer de manière sécuriser sans avoir à faire un échange de clé au préalable.

- 1 Bob génère son couple de clés  $(P_k, S_k)$  et publie sa clé publique  $P_k$
- 2 Alice veut communiquer avec Bob. Elle génère une clé de chiffrement symétrique  $K$ .
- 3 Alice chiffre  $K$  avec la clé publique de Bob  $P_k$  et envoie le message chiffré à Bob.
- 4 Bob déchiffre le message d'Alice avec sa clé privée  $S_k$  et se retrouve donc en possession de la clé  $K$ .
- 5 Par la suite, Alice et Bob utilise la clé  $K$  et un algorithme de chiffrement symétrique pour communiquer.

# Signature numérique : définition

## Définition : signature numérique

Une signature numérique est un mécanisme cryptographique utilisé pour authentifier l'intégrité et l'origine d'un message, d'un document électronique ou d'un ensemble de données.

Elle sert à garantir qu'un document n'a pas été altéré depuis sa signature et qu'il provient bien de l'expéditeur prétendu.

La signature numérique repose sur des algorithmes de cryptographie asymétrique, qui utilisent une paire de clés : une clé privée et une clé publique.

L'expéditeur utilise sa clé privée pour générer une signature numérique unique basée sur le contenu du document. Cette signature est ensuite jointe au document, qui peut être transmis à d'autres parties.

# Signature numérique : exemple

- ① Alice génère sa paire de clé publique/privée  $(P_k, S_k)$
  - ② Alice publie sa clé publique  $P_k$
  - ③ Alice veut signer le message « Alice donne 1 BTC à Bob »
    - ① Alice calcule le hash de « Alice donne 1 BTC à Bob »
    - ② Alice chiffre le hash avec sa clé privée et diffuse le message+le hash signé
  - ④ Tout individu peut maintenant vérifier que le message qu'Alice a chiffré est bien le hash du message qu'elle a publié
- ⇒ Alice a **signé** le message « Alice donne 1 BTC à Bob »



# Introduction à la blockchain

# Sommaire

- 1 Cryptographie
- 2 Introduction à la blockchain
  - Définitions générales
  - Exemple du Bitcoin
- 3 Blockchain Ethereum et smart contracts

# Objectifs de ce module

- ① Comprendre la définition de blockchain en tant que système
- ② Découvrir Bitcoin et le vocabulaire de la Blockchain
- ③ Comprendre le problème du consensus et comment Bitcoin le résoud

# Définitions générales

# Contexte historique : origines de la blockchain

- 2008 : Satoshi Nakamoto publie « Bitcoin: A Peer-to-Peer Electronic Cash System »
- Dans ces neuf pages, Nakamoto décrit un système financier et introduit les bases de la blockchain
  - Structure en blocs
  - Cryptographie (hachage, asymétrique, arbres de Merkel...)
  - Transactions
- Fun fact : Satoshi Nakamoto est toujours resté anonyme

## Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto  
satoshin@gmx.com  
www.bitcoin.org

**Abstract.** A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Figure – Bitcoin Whitepaper by S. Nakamoto  
mewo  
INFORMATIQUE

# Définition générale

Blockchain se traduit par « chaîne de blocs ». Il s'agit donc d'un système permettant de stocker et de partager de l'information au travers d'un **structure de données bien choisie construite à partir de plusieurs blocs** (et c'est tout).

La majorité des systèmes de blockchain possèdent des caractéristiques supplémentaires qui sont utilisées par abus de langage :

- ❶ Présence d'une cryptomonnaie liée à la blockchain (il existe des blockchains SANS cryptomonnaies)
- ❷ Décentralisation
- ❸ Autonome/sans administration centrale
- ❹ Anonymat/pseudonymat des utilisateurs

# Définitions tierces

economie.gouv.fr

*Développée à partir de 2008, c'est, en premier lieu, une technologie de stockage et de transmission d'informations. Cette technologie offre de hauts standards de transparence et de sécurité car elle fonctionne sans organe central de contrôle.  
Plus concrètement, la chaîne de blocs permet à ses utilisateurs - connectés en réseau - de partager des données sans intermédiaire.*

Wikipédia

*Une blockchain, ou chaîne de blocs, est une technologie de stockage et de transmission d'informations sans autorité centrale. Techniquement, il s'agit d'une base de données distribuée dont les informations envoyées par les utilisateurs et les liens internes à la base sont vérifiés et groupés à intervalles de temps réguliers en blocs, formant ainsi une chaîne.*

# Centralisation

Exemples :

- ❶ L'Euro : la banque centrale européenne est souveraine et peut émettre des euros
- ❷ La force nucléaire en France : contrôlée par l'armée
- ❸ Twitter : la direction peut décider de retirer des privilèges sans l'approbation des utilisateurs (arrivée d'Elon Musk...)

- ⇒ La centralisation place un privilège/pouvoir entre les mains d'un petit groupe
- ⇒ Inversement, les utilisateurs sont tributaire du bon vouloir/bon fonctionnement des systèmes
- ⇒ Une relation de **confiance** est nécessaire



# Exemple du Bitcoin

# Bitcoin : décentralisation

- La blockchain Bitcoin est un réseau peer-to-peer décentralisé
- Le réseau Bitcoin **toujours en ligne** (tant qu'il y a des noeuds)
- Pas d'administration centrale (donc pas de Bitcoin Corp. Limited)
- Tout individu peut y participer en créant un « nœud » = démarrer un logiciel en CLI

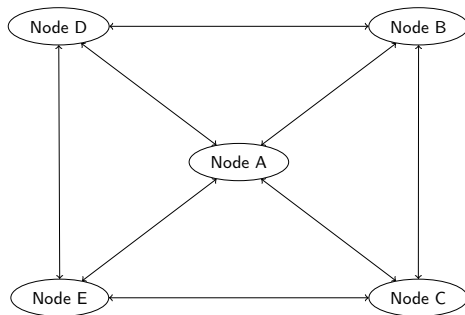


Figure – Réseau peer-to-peer

# Bitcoin : livre de comptes

La blockchain Bitcoin est un système décentralisé permettant aux utilisateurs d'échanger une monnaie numérique, le Bitcoin.

- Les Bitcoin (BTC) sont stockés dans des comptes, identifiés par une adresse
- L'ensemble des soldes des comptes, le « livre de comptes » est stocké à de multiples endroits
- Tout individu peut obtenir un compte gratuitement (on en parle plus tard)
- Envoyer des  $x$  BTC d'une adresse  $a$  à une adresse  $b$  revient à faire

```
1  solde_a -= x; solde_b += x
```

Compte	Solde
0001	12
0002	3.42
0003	4.4
0004	3.6
0005	5
⋮	
1232	30.45
1233	0.34
1234	113.3
1235	4.97

# Bitcoin : opérer un node

- Opérer un node = participer à la blockchain = augmenter la décentralisation
- « Relativement léger » : 2 Go de RAM, 7 Go de disque, connexion 400 kilobits/sec
- Attention, certains pays interdisent d'opérer un node : Afghanistan, Algérie, Bangladesh, Bolivie, Chine, Égypte, Kosovo, Maroc, Népal

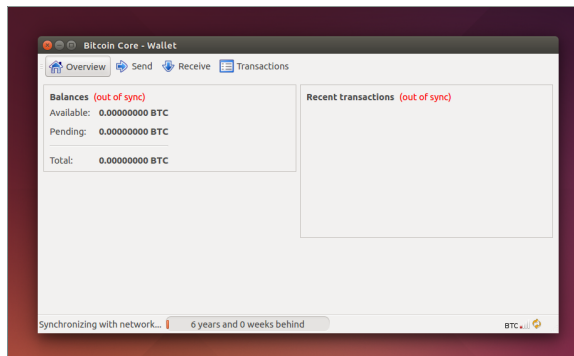


Figure – Bitcoin Core GUI

# Bitcoin : créer un compte

- La blockchain Bitcoin, comme la majorité des blockchain, fonction avec un couple de clé publique/privée.
- Chaque clé privée donne accès à un compte (=adresse) de la blockchain.
- Attention, si la clé privée fuite, le compte est définitivement compromis.

Exemple avec <https://iancoleman.io/bip39/> :

- Clé privée Kwp24SX6Uo1xNUDihM3oeSmg9MHrPm9ZEp7v26jrmxMmC5js4i5f
- Clé publique  
0256fa0a8c520a0a501000845ebaf112295b3c5c29b4b0f7b2d01b933451d9ebf8
- Adresse 1CLHPJK9Z1NFzvQSnP5CozanMo5L35Caq1

# Bitcoin : effectuer une transaction

## Définition : transaction

Sur la blockchain Bitcoin, une transaction est un message signé (avec la clé privée) par un utilisateur de blockchain. Elle contient les informations suivantes :

- L'adresse de l'émetteur.
- L'adresse du destinataire.
- La valeur (=montant) de la transaction.

# Bitcoin : effectuer une transaction

Exemple : Alice, qui possède 10 BTC, peut en envoyer 1 BTC à Bob en signant le message suivant : « Moi, Alice, envoie 1 BTC à Bob »

Une fois la transaction signée, il faut l'envoyer au réseau Bitcoin. Pour cela, Alice doit envoyer la transaction à un node Bitcoin.

Ce dernier va alors vérifier la transaction (si la signature est valide) et si c'est le cas, l'envoyer à d'autres noeuds du réseau, qui vont eux-mêmes vérifier la transaction, etc. Par effet boule de neige, la transaction fini par atteindre rapidement tous les noeuds Bitcoin.

# Bitcoin : effectuer une transaction

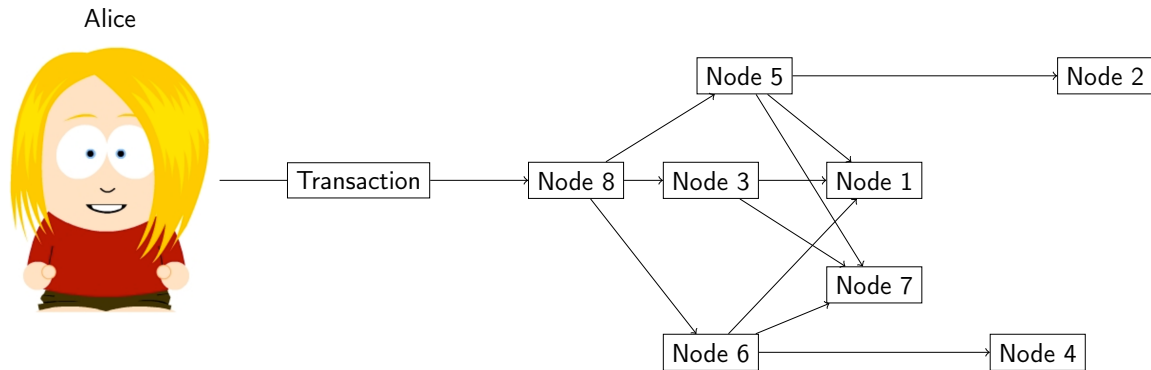


Figure – Diffusion ou « broadcast » d'une transaction



# Bitcoin : problème du consensus

- Dans le schéma précédent, Alice a initialement envoyé sa transaction au Node 8, ce qui implique que celui-ci a eu temporairement une liste de transactions différente des autres nodes.
  - En parallèle, d'autres individus envoient des transactions à d'autres nodes.
  - En pratique, aucun node n'a exactement la même liste de transactions.
- ⇒ C'est le problème du consensus = dans un système décentralisé, comment faire en sorte de mettre tout le monde d'accord ?

# Bitcoin : structure en blocs

## Définition : bloc

Dans une blockchain, un bloc est un ensemble de transactions.

- Dans la blockchain Bitcoin, la taille d'un bloc est limité à 1 MB ce qui correspond à environ 2000 transactions.

```
1  Block #4598794
2
3  Alice sends 10 BTC to Bob
4  John sends 5 BTC to Alfred
5  Juliette sends 3 BTC to Mathieu
6  Bob sends 3 BTC to Mathieu
7  Mathieu sends 4 BNTC to John
8
9  Nonce 90348953a8df7fe9
```

Figure – Exemple textuel d'un bloc

# Bitcoin : consensus par preuve de travail « proof-of-work »

- Afin d'obtenir le consensus, Satoshi Nakamoto a choisi d'utiliser la preuve de travail ou « proof-of-work ».
- Afin de proposer son bloc, le node doit résoudre un problème cryptographique difficile : il doit réussir à concevoir un bloc dont le hash avec l'algorithme SHA-256 commence par un certain nombre de zéro.
- Pour cela, le node fait varier le **nonce** du bloc, qui est un text arbitraire ajouté en fin de bloc.

```
1  Block #4598794
2
3  Alice sends 10 BTC to Bob
4  John sends 5 BTC to Alfred
5  Juliette sends 3 BTC to Mathieu
6  Bob sends 3 BTC to Mathieu
7  Mathieu sends 4 BNTC to John
8
9  Nonce 90348953a8df7fe9
```

Figure – Exemple textuel d'un bloc

# Bitcoin : consensus par preuve de travail « proof-of-work »

- Si le node trouve la solution, il doit la diffuser au reste du réseau qui vérifieront le nonce trouvé.
- En récompense, le node reçoit 6.25 BTC soit environ 150000€ au 14 mai 2023.
- La seule manière de trouver la solution est la force brute (cf. sécurité des fonctions de hachage).
- On appelle également les nodes « mineurs ».
- Un bloc est miné toutes les 10 minutes.

```
1  Block #4598794
2
3  Alice sends 10 BTC to Bob
4  John sends 5 BTC to Alfred
5  Juliette sends 3 BTC to Mathieu
6  Bob sends 3 BTC to Mathieu
7  Mathieu sends 4 BNTC to John
8
9  Nonce 90348953a8df7fe9
```

Figure – Exemple textuel d'un bloc

# Bitcoin : aspect économiques et halving

- Sachant que chaque bloc miné introduit des nouveaux bitcoin dans le marché, la valeur du bitcoin baisse nécessairement sur la durée (plus de bitcoin=moins de valeur pour 1 bitcoin).
- Ce phénomène est comparable à l'inflation des monnaies traditionnelles.

## Halving (de l'anglais *half*, moitié)

Dans la blockchain Bitcoin, les récompenses de bloc sont divisées par deux tous les 210000 blocs, soit environ 4 ans. En 2009, un bloc valait 50 BTC, en 2013 25 BTC, en 2016 12 BTC et en 2020 6.25 BTC.

## Maximum supply

Dû au mécanisme de halving, il ne pourra jamais y avoir plus de 21 millions de BTC en circulation. Aucun État, organisation, armée ou puissance ne pourra en décider autrement.

# Bitcoin : minage

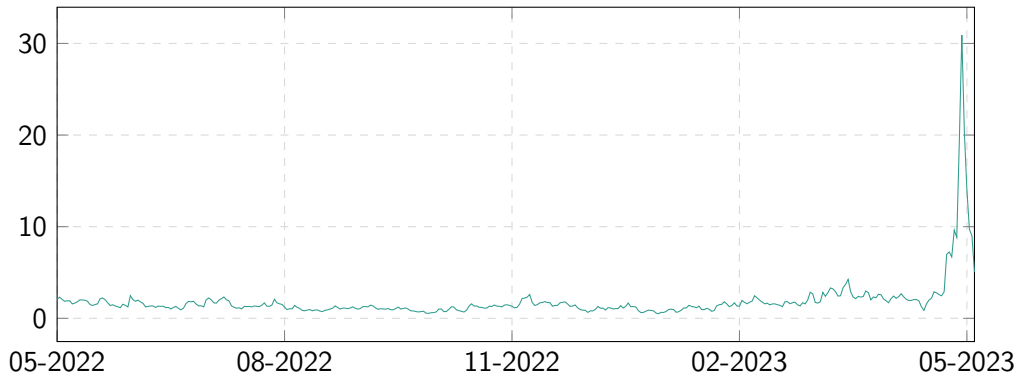
- Vu les récompenses de blocs, le minage s'est rapidement professionnalisé.
- Des entreprises ont créé spécifiquement désigné pour le minage du Bitcoin, 20 fois plus puissants que les meilleurs processeurs AMD/Intel.
- La consommation électrique du minage de Bitcoin atteint 140 TWh par en mars 2023 (Univ. Cambridge), soit l'énergie consommé par l'Égypte ou la moitié du parc nucléaire français.



Figure – Un mineur Antminer L7

# Frais de transaction

Émettre une transaction sur une blockchain n'est jamais gratuit et se paye en BTC.



# Blockchain Ethereum et smart contracts



# Sommaire

- 1 Cryptographie
- 2 Introduction à la blockchain
- 3 Blockchain Ethereum et smart contracts**

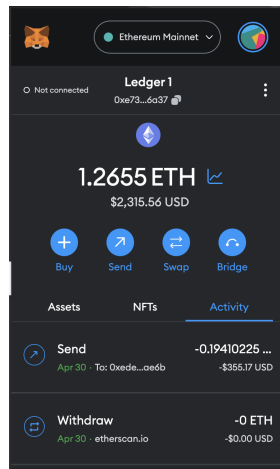
# Objectifs de ce module

Comprendre les opérations de base de la cryptographie

- 1 Interagir avec une blockchain de test (Ethereum Sepolia)
- 2 Interagir avec une blockchain de production (Polygon)

# Créer un wallet avec MetaMask

- Ceux qui on déjà un wallet : bonne sieste
- Les autres, installez l'extension MetaMask sur votre navigateur (Chrome / Firefox)
- Effectuez l'onboarding et notez bien la phrase de récupération



# Seed phrase

Lors de la création d'un wallet avec MetaMask, une phrase de douze mots est générée.

Il faut absolument la sauvegarder en lieu sûr.  
Quiconque l'obtient peut accéder à tout votre  
wallet, vos cryptos et vos NFTs !

# Foundry

# Foundry

Dans ce cours, nous allons utiliser Foundry, un utilitaire permettant de développer des smart-contracts simplement.

# Installation de Foundry : Windows

Il faut d'abord installer Windows Subsystem Linux avec :

```
1 $ wsl --install
```

WSL demandera de redémarrer, puis au redémarrage de choisir un nom d'utilisateur et un mot de passe. Pour confirmer la bonne installation de WSL, exécuter :

```
1 $ wsl --list
```

# Installation de Foundry

Configurer git avec votre nom / email :

```
1 $ git config --global user.email "you@example.com"
2 $ git config --global user.name "Prénom Nom"
```

Installer Foundryup (installateur de Foundry)

```
1 $ curl -L https://foundry.paradigm.xyz | bash
```

Redémarrer le terminal, lancer Foundryup

```
1 $ foundryup
```

Redémarrer le terminal, lancer Forge

```
1 $ forge
2 forge 0.2.0 (31fcf5a 2023-05-19T00:10:33.861185000Z)
```



# Architecture d'un projet Foundry

/	
├── out.....	Fichiers compilés
├── lib.....	Libraries installées
├── src.....	Code source des contrats
├── test.....	Test des contrats
├── .gitmodules	
└── foundry.toml.....	Configuration de Foundry

# Solidity

# Qu'est-ce qu'un smart contract

## Définition : smart contract

Sur la blockchain Ethereum, un smart contract est un bytecode (=code hexadécimal) associé à une adresse.

⇒ Les adresses des smart contracts sont indiscernables des adresses des comptes utilisateurs.

## Définition : Externally Owned Account

Les adresses contrôlés par des utilisateurs sont appelées « Externally Owned Account » (EOA).

Voir le glossaire d'Ethereum : <https://ethereum.org/en/glossary>

# Solidity ?

## Définition : Solidity

Solidity est un **langage de programmation** utilisé pour écrire des smart contracts sur la plateforme Ethereum.

Solidity permet aux développeurs de définir des règles et des logiques spécifiques à un smart contract. Il permet d'écrire des lignes de code qui définissent comment un smart contract doit fonctionner, quelles actions il doit effectuer et comment il doit réagir dans différentes situations. En utilisant Solidity, les développeurs peuvent créer des smart contracts pour diverses applications décentralisées (dApps)

# Solidity : syntaxe en POO

```
1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity ^0.8.13;
3
4  contract Counter {
5      uint256 public number;
6
7      function setNumber(uint256 newNumber) public {
8          number = newNumber;
9      }
10 }
```

# Solidity : header

Le header d'un smart contract s'écrit en deux lignes :

```
1  // SPDX-License-Identifier: UNLICENSED  
2  pragma solidity ^0.8.13;
```

La ligne 1 définit la licence du fichier :

- UNLICENSED signifie que le code est complètement privé
- `pragma solidity ^0.8.13;` version de Solidity compatible avec le fichier

# Semantic versionning

## Semantic versionning « SemVer »

Le versionnage sémantique est une méthode de numérotation des versions logicielles basée sur des règles spécifiques. Elle se compose de trois nombres séparés par des points : MAJEUR.MINEUR.PATCH. Le numéro MAJEUR est augmenté lorsque des changements incompatibles sont apportés, le numéro MINEUR est augmenté lorsque des fonctionnalités sont ajoutées de manière rétrocompatible, et le numéro PATCH est augmenté pour les corrections de bugs rétrocompatibles.

### Opérateurs

- $=1.2.3$  strictement égal à 1.2.3
- $\wedge 1.2.3 \Rightarrow 1.2.3 < v < 2.0.0$
- $\sim 1.2.3 \Rightarrow 1.2.3 < v < 1.3.0$

### Opérateurs (MAJEUR=0)

- $=0.1.2$  strictement égal à 0.1.2
- $\wedge 0.1.2 \Rightarrow 0.1.2 < v < 0.2.0$

# Solidity : types primitifs

- `uint` un entier non signé sur 256 bits
- `uint8` un entier non signé sur 8 bits
- `uint32` un entier non signé sur 32 bits
- `uint256` un entier non signé sur 256 bits
- `int` un entier signé sur 256 bits
- `int32` un entier signé sur 32 bits
- `address` une adresse Ethereum
- `string` une chaîne de caractères
- `struct` structure, au sens langage C du terme
- `mapping` une association clé-valeur



# Solidity : le type address

Le type `address` est spécial et possède des propriétés :

- `<address>.balance` le montant d'ether détenu par l'adresse
- `<address>.code` le code à l'adresse (vide pour les EOA)

# Solidity : variables spéciales

## `msg` (= message)

Un message représente un appel d'une fonction d'un smart contract.

- `msg.sender` expéditeur du message
- `msg.value` nombre d'Ether envoyés avec le message

## `block`

Metadata du bloc actuel.

- `block.number` numéro du bloc actuel
- `block.timestamp` timestamp UNIX en secondes

# Solidity : contrôle de flow

**if / else / else if**

```

1  if (cond) {
2    // if path
3  } else if (cond2) {
4    // else if path
5  } else {
6    // else path
7  }
```

Note : pas de switch / case en Solidity.

**for / while**

```

1  for (uint256 i = 0; i < n; i++) {
2    // do stuff
3  }
4
5  while(cond) {
6    // do other stuff
7  }
```

Note : pas de **do while** en Solidity.

# Solidity : visibilité

## Public

```

1  uint public myVariable;
2  function myFunction() public {
3      // Function logic
4  }
```

## Private

```

1  uint internal myVariable;
2  function myFunction() internal {
3      // Function logic
4  }
```

## Internal

```

1  uint internal myVariable;
2  function myFunction() internal {
3      // Function logic
4  }
```

## External

```

1  // external variables not possible
2  function myFunction() external {
3      // Function logic
4  }
```

# Exemple : cryptomonnaie

```
1  contract Mewo {
2      mapping(address => uint256) public balances;
3
4      function transfer(address to, uint256 amount) public {
5          require(balances[msg.sender] >= amount, "Insufficient balance");
6          balances[msg.sender] -= amount;
7          balances[to] += amount;
8      }
9  }
```

# Solidity : interfaces

Comme beaucoup d'autres langages, Solidity dispose d'interfaces<sup>1</sup> qui servent à intégrer des notions de polymorphisme.

- Elles ne peuvent pas hériter d'autres contrats, mais elles peuvent hériter d'autres interfaces.
- Toutes les fonctions déclarées doivent être externes.
- Elles ne peuvent pas déclarer de constructor, de variables d'état ou de modifier.

## Syntaxe : interface

```
1 interface IToken {  
2     function transfer(address recipient, uint amount) external;  
3 }
```

---

1. Voir la documentation Solidity des interfaces

# Solidity : erreurs

Souvent, il faut arrêter l'exécution d'un smart contract et renvoyer une erreur (exécution non autorisée, opération impossible, etc.).

Fonction `revert(bool assertion, string message)`

Revert si `assertion` est évalué à `false`.

Custom errors : permet de déclarer des erreurs avec des paramètres.

```
1  contract Foo {  
2      error Custom(uint256 arg1);  
3  
4      function willRevert() public {  
5          revert Custom(1);  
6      }  
7  }
```

# Exemple : cryptomonnaie avec mint initial

```
1  contract Mewo {
2      uint256 constant MAX_SUPPLY = 10000000000; // 1 billion
3      mapping(address => uint256) public balances;
4
5      constructor() {
6          balances[msg.sender] += MAX_SUPPLY; // Initial mint
7      }
8
9      function transfer(address to, uint256 amount) public {
10         require(balances[msg.sender] >= amount, "Insufficient balance");
11         balances[msg.sender] -= amount;
12         balances[to] += amount;
13     }
14 }
```



# Exemple : cryptomonnaie avec mint

```
1  contract Mewo {
2      mapping(address => uint256) public balances;
3
4      function mint(uint256 amount) public {
5          balances[msg.sender] += amount
6      }
7
8      function transfer(address to, uint256 amount) public {
9          require(balances[msg.sender] >= amount, "Insufficient balance");
10         balances[msg.sender] -= amount;
11         balances[to] += amount;
12     }
13 }
```

# Solidity : modifier

## Définition : modifier

En Solidity, un « modifier » est une fonction spéciale qui permet de modifier le comportement d'autres fonctions dans un contrat intelligent. Les modifieurs fournissent un moyen pratique de réutiliser du code et d'ajouter des conditions supplémentaires ou des vérifications avant l'exécution d'une fonction.

## Syntaxe : modifier

```
1  modifier exampleModifier() {  
2      _; // Continue function execution  
3  }  
4  
5  function foobar() public exampleModifier {}
```

# Exemple : cryptomonnaie avec mint protégé

```
1  contract Mewo {
2      address owner;
3      mapping(address => uint256) public balances;
4
5      constructor() {
6          owner = msg.sender;
7      }
8
9      modifier onlyOwner() {
10         require(msg.sender == owner, "Only owner");
11         _;
12     }
13
14     function mint(uint256 amount) public onlyOwner {
15         balances[msg.sender] += amount;
16     }
17 }
```

# Notion de gas

- Les frais Ethereum ne paie pas à la transaction mais à la **complexité du calcul**
- Transférer de l'Ether entre deux comptes est beaucoup moins coûteux que faire participer à une enchère de NFTs
- La complexité des transactions en Ethereum se mesure en « gas »
- Le prix d'un « gas » se mesure en Gwei avec  $1 \text{ ETH} = 1000000000 \text{ Gwei}$ .
- L'utilisateur peut choisir le prix du « gas » affecté à sa transaction.

## Exemple : Mewo.mint

- Gas utilisé pour la fonction mint : 24634
- Prix du gas : 37 gwei/gas
- Prix d'un ETH = \$1,817.85
- Total =  $24634 \times 37 = 911458 \text{ gwei} = 0.000911458 \text{ ETH} = \$1.657$