# Fundamentals of Cryptography

## Introductory Lecture Series

Rhys Bidder
rhys.m.bidder@kcl.ac.uk

KBS, QCGBF

September 2023

*Disclaimer 1:* Any opinions expressed, errors or omissions should be regarded as those of the author and not necessarily those of KCL, KBS, QCGBF, QCB or BoE.

*Disclaimer 2:* These notes on cryptography are heavily simplified and leave out many important details. For any real-world security applications these notes should not be relied upon. Instead you should consult appropriate security resources and reliable security professionals.

# Outline

**Cryptoasset transactions are illegal in many jurisdictions. Do not violate these or any other laws. I am not promoting the use of crypto in countries where it is illegal in any form and these slides are not a promotion of crypto or an invitation to participate in crypto-related activities in such countries. They are purely for educational purposes.**

# Introduction

# Introduction

- The prefix 'crypto' in 'crypto-assets' reflects the fact that their design depends heavily on results and techniques from the area of Mathematics called **cryptography**
- Cryptographic techniques will recur *over and over again* in our discussions...
  - ...and in press, commercial, policymaker and academic debates
  - While they may be challenging at first sight, they can't be avoided!
- In this lecture we go through key concepts to provide a foundation for the rest of the course
  - We will frequently refer back to these notes

# Hashing

'Hashing' data is a very boring but vitally important element of blockchain systems

- The goal is to compress any data to a (very) compact representation of fixed length that is easy to store, reference and compare against other hashes
- Loosely speaking, it doesn't matter how long the input data is, the length of the output is fixed
- For message $m$ and hash function $\mathcal{H}$ we generate hash $M$

$$M = \mathcal{H}(m)$$

# Hashing

In a crypto context the hash function should be:

- **Quick**
  - Though perhaps not too quick
- **Deterministic**
  - Different hashes $\Rightarrow$ Different input
- **Pseudorandom**
  - No pattern to how changing input changes the hash
- **One-way**
  - Knowing the output of the hash alone doesn't reveal the input
- **Collision resistant**
  - *Effectively* zero chance of different inputs $\Rightarrow$ same output

# Hashing

Let's try some hashing...

- Website associated with (excellent) Blockchain Basics book
  - I will draw upon this book (and others) during these lectures
  - See bibliography of suggested readings for other references
- Focusing on SHA-512 we use this website to explore further
  - Consider a weak password - like *mypassword*
  - Consider a stronger password - like *7Hj(;gD3e:*

# Hashing

- **Q:** Why was the weak password broken, given that the hash has the ‘one way’ property?

# Hashing

- **Q:** Why was the weak password broken, given that the hash has the 'one way' property?
- **A:** Because there are dictionaries of hashes for common phrases (such as *mypassword*) **and** of passwords revealed in historical breaches
  - Simply look up the hash in the dictionary
  - Collision resistance $\Rightarrow$ guaranteed to reveal the input
- **Implication:** Pick long and complicated passwords!
  - Humans are bad at choosing randomly
  - Good practice to use a (high quality, expert-developed) random password generator

▸ Salt

# Hashing

A key property that is vital for functionality of blockchains

- Easy to detect changes in data

# Hashing

A key property that is vital for functionality of blockchains

- Easy to detect changes in data

Suppose I write a message ‘Meet me in Doha’ and file it away

- Now suppose, while I’m not looking, someone changes it to ‘Meet me in London’
- When I see the changed version, the tampering is *immediately* and *easily* detectable

# Hashing

SHA-512('Meet me in Doha')

*1dc033934f9b898271790e3dffd6cd6258c804b81f0e09517aca97d67e8f45fd6894936e7767bd3ff1464a643dddc491a4674fc43b866ac0ba44a6dd996722fb*

SHA-512('Meet me in London')

*1caa192b916a5257cca20c05babc093400c99b6a46764224a11dd6adeaca8731fa8ca41ae60b863b9b0a4d29733594fb73cd794eb24cba45223b52d9f8fe63d5*

# Hashing

SHA-512(<entire text of Tolstoy's War and Peace>)

*278d75e313af4bb8ff3ab2da3307c22a15c61f4bc5b0fadf5389129838eb0402c29379853f6762ab63a9a26bc55dda892b03006ce4232706cb19a6ef601462bd*

SHA-512(Amended <entire text of Tolstoy's War and Peace>)

*b5dbb25cbbe1786e45f153a10ff8713193816fc5e5d0cad51058bc5af60ddaa25f32c93b38000dea0ae0e8ee5e0c4ae6dfe40ad1fd74008d2e46e6e9407d6cea*

**Note:** Creating the hashes was almost instantaneous

# Hashing

Original

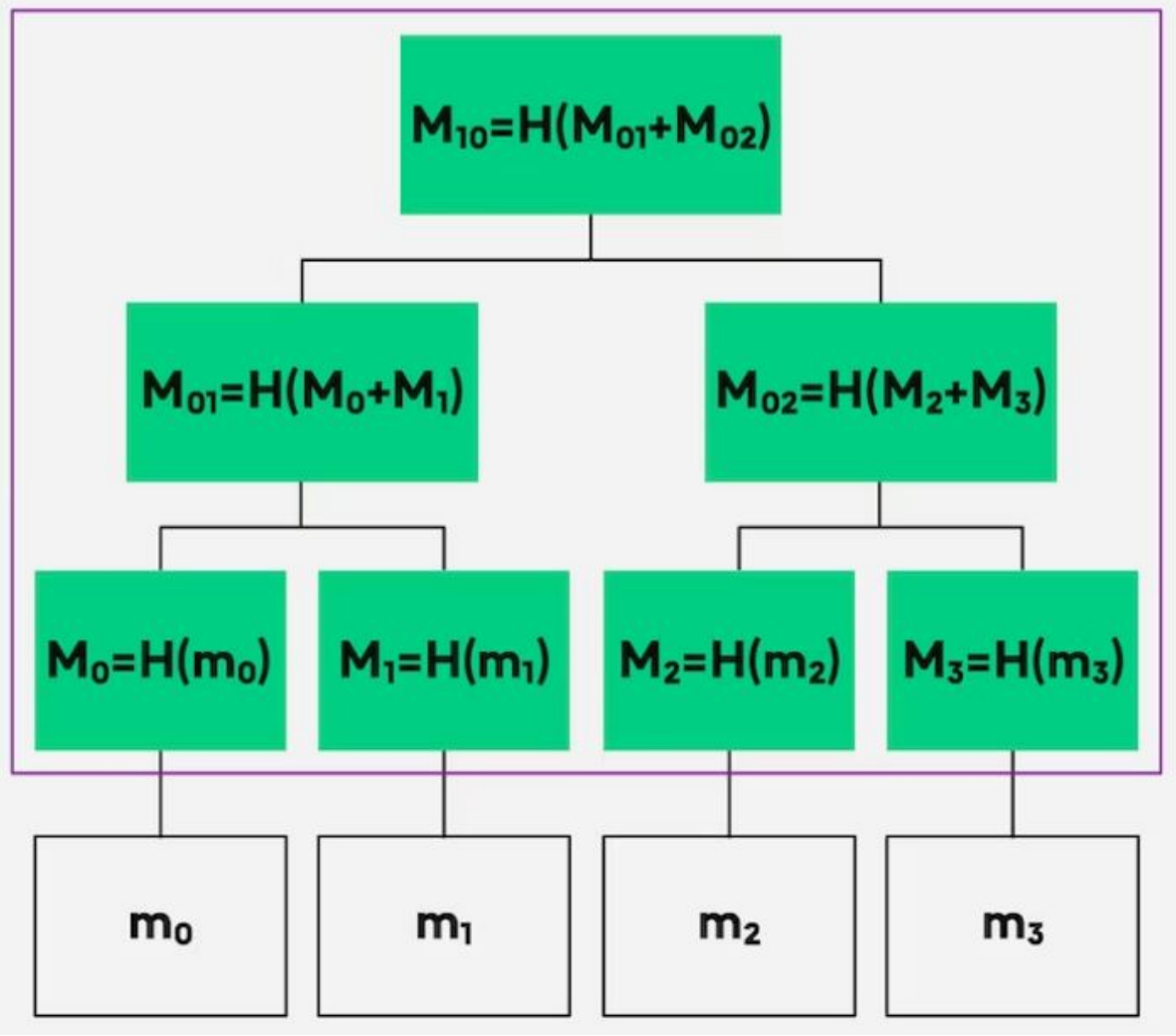
> *All we can know is that we know nothing. And that's the height of human wisdom.*

Amended

> *A1l we can know is that we know nothing. And that's the height of human wisdom.*

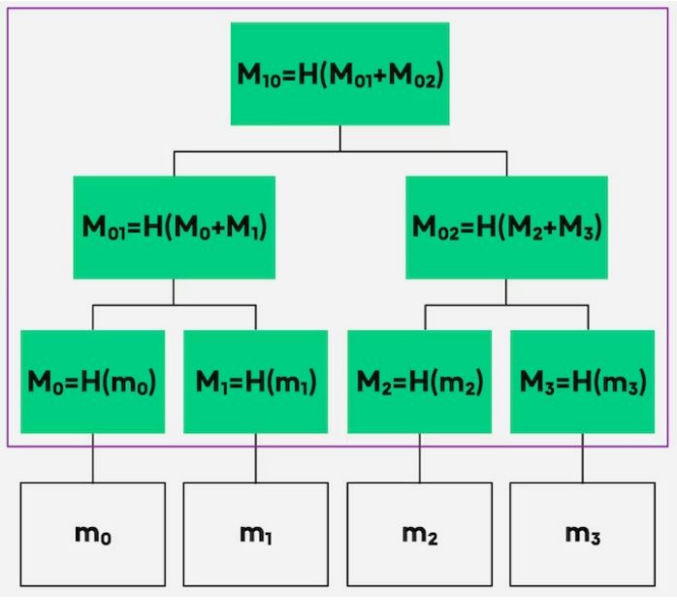
**Note:** Transmitting and comparing hashes is useful!

# Merkle trees




Source: Bitpanda - Everything you need to know about Merkle trees

# Merkle trees




- Original data $(m_0, m_1, m_2, m_3)$ are not part of the tree - their hashes are $(M_0, M_1, M_2, M_3)$
- Moving up tree: pairs of hashes concatenated and then, themselves, hashed
- Process continues up the tree until the 'root' hash

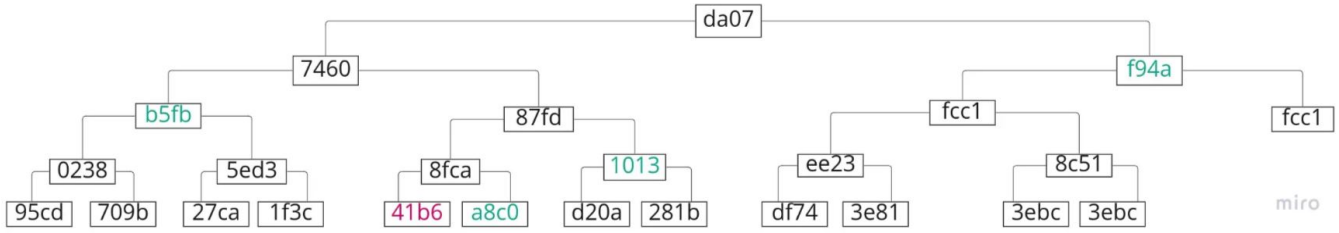
**Silly:** Always drawn so the **root** is at the **top** of the tree!

# Merkle trees

Suppose we want to (quickly) check if a given piece of data is included in the underlying set of data, captured in the Merkle tree?

- We only need a small subset of nodes from the tree (a Merkle 'proof' or 'path')
- *Dramatically* speeds up search relative to checking list of (the hashes of) all data inputs

# Merkle trees




Source: Medium - Merkle Tree, a simple explanation and implementation.

**Q:** Is 41*b*6 in the dataset with root hash *da*07?

- Only additionally need to know $\{a8c0, 1013, b5fb, f94a\}$
- 'Climb' tree, hashing relevant nodes $\Rightarrow$ candidate root hash
- If doesn't match *da*07 then 41*b*6 is not present
- Imagine: Replace 41*b*6 with 6*hhg* - what would happen/why?

**If any data in the tree is changed, it can be detected extremely quickly.**

**If data exists in the underlying set, it can be located quickly.**

# Merkle trees

Why are they useful?

- Compact representation of data
- Equality of two sets of underlying data can be established just by comparing root hashes
- Source of differences can be rapidly identified - can stop searching down a sub-tree if hashes match
- Structure amenable to parallel processing - searching one sub-tree can be done independently of searching another
- Efficient algorithms to check presence of a given piece of underlying data

**Transactions will be one type of data being hashed in our applications and Merkle roots will feature prominently**

# Cryptography

Cryptography:

- **Encryption:** Converting plaintext into cyphertext
- **Decryption:** Recovering the plaintext from cyphertext

Cyphertext should be incomprehensible to anyone other than the intended reader, who is able to decrypt it back to plaintext

- Various algorithms exist
- Many of them involve a key (or multiple keys)
- Even if the algorithm is precisely known by an attacker, a (secure) system will not be broken without knowledge of the key or keys

# Symmetric key

One approach is 'symmetric key cryptography'

- Both parties (Abia and Bilal) know the same key, $K$
- Abia encrypts plaintext, $P$, with a function that depends on the key, $\mathcal{E}_K$, and sends the cyphertext, $C$, to Bilal
- Bilal decrypts $C$ using a function, $\mathcal{D}_K$, that is, conditional on $K$, the inverse of $\mathcal{E}_K$

$$\begin{aligned} C &= \mathcal{E}_K(P) \\ P &= \mathcal{D}_K(C) = \mathcal{D}_K(\mathcal{E}_K(P)) \end{aligned}$$

Thus $\mathcal{D}_K = \mathcal{E}_K^{-1}$

# Symmetric key

What's the problem?

- The 'key transfer' problem - how do Abia and Bilal **both** come to know $K$?

Sometimes possible at the start of an operation, but what about after sending an agent into the field?

- Borrowing the language of spy world!
- A lot of traditional use cases came from this domain

# Symmetric key

In the past, elaborate, expensive and possibly insecure methods were used to exchange a symmetric key

- Feasible if you know whom you need to contact
- But what if Abia doesn't even know (or will never know) Bilal?

**In decentralized blockchain systems like Bitcoin/Ethereum, no one knows anyone!**

# Symmetric key

Simple alphabetic shift cipher

- **Plaintext: a test string**
- **Cyphertext: f yjxy xywnsl**

What was the key?

# Symmetric key

Simple alphabetic shift cipher

- **Plaintext: a test string**
- **Cyphertext: f yjxy xywnsl**

What was the key? 5

Obviously, this is a horrible protocol

- I coded this up in Mathematica but it is simple with pen and paper too

# Asymmetric key: Encryption

In asymmetric key systems, Abia and Bilal *each* possess two keys - a private key and a public key derived from the private key

- **Abia:** Has $K_A$ (private) and $k_A$ (public)
- **Bilal:** Has $K_B$ (public) and $k_B$ (public)

Abia and Bilal each make their public key known publicly (hence the name)

- The public key is derived from the private key using a method that cannot feasibly be reversed in any reasonable time
- **Private keys must always remain private** (at least until the need for encryption has expired)

# Asymmetric key: Encryption

- Bilal sends $k_B$ to Abia
  - Indeed, he could broadcast $k_B$ to everyone
  - Will return to this when we discuss Certificate Authorities
- Abia encrypts a message $m$ using $k_B$ and sends it to Bilal

$$C = \mathcal{E}_{k_B}(M)$$

- **$C$ can only be decrypted with knowledge of $K_B$ which only Bilal is assumed to have**

$$m = \mathcal{D}_{k_B, K_B}(C)$$

**Key exchange problem is 'solved' - different (but related) keys are used for encryption and decryption**

# Asymmetric key: Encryption

- Is it magic?
  - No, but it is brilliant.
- Some issues...
  - More computationally intensive
  - Size($C$) $\geq$ Size($P$) whereas Symmetric $\Rightarrow$ Size($C$) $\leq$ Size($P$)
  - Longer keys required
  - Still has vulnerabilities (e.g. 'man-in-the-middle' attack)
- Slower but more secure than symmetric case
  - Approaches can be combined
  - Use AKC to share a *symmetric* key and then use that for SKC
  - Allows large amounts of data to be exchanged quickly

# Asymmetric key: Encryption

Man-in-the-Middle attack is a vulnerability in naive AK setups:

- Suppose Eve knows Abia wants to send a message to Bilal, using AKC, and that he is expecting it
- Eve creates a private-public key pair ($K_{EB}$ and $k_{EB}$)
- Eve sends $k_{EB}$ to Abia, **pretending to be Bilal**
- Abia uses $k_{EB}$ to encrypt $m$ to $C$
- Abia sends $C$ to Eve, **thinking it's Bilal**
- Eve decrypts $C$ and can read $m$ (or change to $m'$)
- Eve then encrypts $m$ (or $m'$) using $k_B$
- Eve sends ciphertext to Bilal who decrypts it with $K_B$

Clearly, Eve could eavesdrop on the *entire conversation* if she also does the reverse for messages from Bilal to Abia

# Asymmetric key: Encryption

**Q:** Any idea how to 'solve' this problem?

# Asymmetric key: Encryption

**Q:** Any idea how to 'solve' this problem?

- Use digital certificates, administered by a Certificate Authority
- Certificates and certificate authorities (CA) are components of a Public Key Infrastructure (PKI)
- The CA - which must be a very reliable and secure institution - confirms the correct identity of an entity
- Provides a *digitally signed* certificate containing various information about the entity **and its public key**
- When counterparties receive a public key via a signed certificate they can be (more) confident in the identity of the entity they are communicating with

**The CA can issue and revoke certificates - thus, it can control who can participate in the secured communications**

**Note this implies a centralization of authority**

# Asymmetric key: Digital signatures

What are digital signatures?

- A flip side of the AKC methods we've already discussed
- Used even where contents of a 'document' are not secret, but where it is important to know that:
  - the claimed signer and only the claimed signer, signed the document and...
  - the document hasn't since been tampered with (i.e. it is what the signer signed off on)

# Asymmetric key: Digital signatures

Suppose Bilal sends Abia a document $d$ for her to sign and return:

- She encrypts the hash of the document with *her private key* to obtain cypher text $C$
- $C$ is sent, with $d$, to Bilal
- Bilal then decrypts $C$ with *Abia's public key*
- Bilal hashes $d$ and checks it matches the decrypted hash
- If so, the document must have been signed by Abia

# Asymmetric key: Digital signatures

Contrast digital signature process (verifying signer) vs. previously discussed encryption (rendering a document secret)

- For digital signatures, the sender encrypts a document with *her private key*
  - In contrast, when discussing encryption, the sender encrypts a document with the *receiver's public key*
- For digital signatures, the receiver decrypts with the *sender's public key*
  - In contrast, when discussing encryption, the receiver decrypts with *his private key*

**Absolutely vital the sender keeps her private key secret, otherwise anybody could sign documents on her behalf.**

# Asymmetric key: Digital signatures

We can *combine* the two aspects of AKC:

- The document could first be sent in encrypted form by Bilal, to be decrypted with Abia's private key
- Abia sends back an encrypted version of the document (to be decrypted by Bilal's private key) along with the signed document
- The document is never sent as plaintext

**Frequently a 'message' to be signed is confidential and should be known only to the counterparties.**

# Zero knowledge proofs

*Perhaps the most powerful cryptographic technology to come out of the last decade is general-purpose succinct zero knowledge proofs*

*For example, you can make a proof for the statement "I know a secret number such that if you take the word 'cow', add the number to the end, and SHA-256 hash it 100 million times, the output starts with 0x57d00485aa. The verifier can verify the proof far more quickly than it would take for them to run 100 million hashes themselves, and the proof would also not reveal what the secret number is.*

Vitalik Buterin, January 26, 2021

# Zero knowledge proofs

Famous example of an 'interactive' zero knowledge proof:

- There is a cave where, soon after the entrance, the path forks
- The two forks, left and right, loop round to form a ring shape
- The paths loop meet at a door
- The door can only be opened with a secret password
- Peggy (the **prover**) wants to prove to Victor (the **verifier**) that she knows the password without
  - revealing the password, or
  - revealing her knowledge of the password to anyone else

# Zero knowledge proofs

- They repeatedly go to the entrance of the cave
- Victor closes his eyes and covers his ears
- Peggy takes either the left or right path
- Victor opens his eyes, uncovers his ears and tosses a fair coin
- If it's heads, he shouts Left, if tails, he shouts Right
- Peggy has to come back via the path he calls
- They do this 'many' (N) times

# Zero knowledge proofs

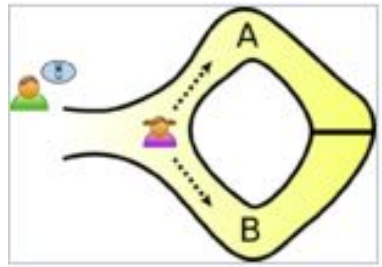
If Peggy knows the password, she will always come back on the called path

- If she had gone down that path initially, she would simply turn around and come back
- If she had gone down the other path, she would simply open the door and come back on the called path

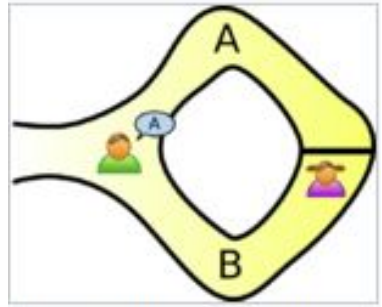
If Peggy does not know the password,

- She might get lucky the first time, maybe even the second. . .
- But the probability of getting it right is 0.5 each time, with each time independent (due to fair coin toss)
- So the probability of her getting it right $N$ times is $0.5^N$
  - $N = 10 \Rightarrow 0.1\%$ chance
  - $N = 20 \Rightarrow 0.0001\%$ chance

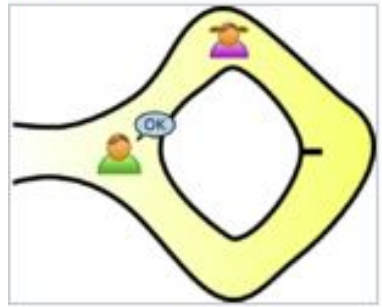
# Zero knowledge proofs




Peggy randomly takes either path A or B, while Victor waits outside




Victor chooses an exit path




Peggy reliably appears at the exit Victor names

Source: Wikipedia

# Zero knowledge proofs

- For large enough $N$, Victor will be convinced
- Even if they record each interaction, other observers will not be convinced
  - If the coin result can't be seen, they might have simply pre-agreed the sequence of called paths (with Peggy going down the paths she knew would be chosen)
  - Even if the coin result is seen, they might have thrown away all the recordings of her failures

# Zero knowledge proofs

Zero knowledge proofs need to satisfy

1. **Completeness:** If the statement is true, an honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover

2. **Soundness:** If the statement is false, no cheating prover can convince an honest verifier that it is true, except with some small probability

3. **Zero-knowledge:** If the statement is true, no verifier learns anything other than the fact that the statement is true

# Zero knowledge proofs

Of course, in our context this is achieved with mathematical methods wildly beyond the scope of this course

- There is currently **enormous** interest - academic and applied - in developing efficient ZKP protocols for blockchain applications
- Offers a promising way to enhance **scalability** and **privacy**
  - **Scalability:** If verifying something takes time and effort, it might be best for one agent to verify it and then simply prove that it has been verified - without all the details being verified again, and without clogging up network communication (which is key in blockchain)
  - **Privacy:** Without identifying yourself, you can prove that you are permitted access. Or, without revealing your account balance, you can prove that you have enough resources to complete a transaction

# Zero knowledge proofs

- To reiterate - ZKP is a very important and very active research area and may be able to replace many other ways of securing privacy without sacrificing efficiency
  - Are beginning to find substantive real world application
- Another frontier technique (too advanced and no time) is homomorphic encryption
  - Allows operations to be applied to data (such as by a smart contract) without decrypting the data
  - Seems like **magic** - arithmetic operations can be done on encrypted inputs, and get the right answer, without decrypting the data

# Summary

- Hashes compress data to a compact, regular form
- Hashes allow quick comparison of complicated data
- Merkle trees are efficient ways of storing (hashes) of data
- Changes to data can be quickly detected by comparing the root Hash of a Merkle tree
- A piece of data can be located quickly within a Merkle tree (if it exists)
- Asymmetric key encryption allows message to be digitally signed, as well as encrypted
- It is absolutely **vital** that private keys remain secret
- ZKP allows truth of a statement to be proved without revealing anything other than the statement's truth

# Escape slides

# Salt

Salting can be used by systems that store passwords:

- Before saving password, append or prepend additional material to the password - called 'salt'
- Salt should be randomly generated (and sufficiently long) whenever a new password is created
- The hash of the password *concatenated with the salt* is then stored with the salt (e.g. by a website admin)
- When a user submits their password, the salt is again concatenated and compared to the stored hash
- If they match then the password is known to be correct

# Salt

SHA-512 hash of *7Hj(;gD3e:* is:

*3793643444be9c4d48139a886a46b8bbda1e4635a1478d791239aa148399e3f...*

*0a1e35bfd1bbced73d0fb8e684eae06c9fe7ce082f60e578b07b08f646e45af74*

Prepend with salt *JHk6hk* and re-hash:

*9cc5b2c6ea470046a30d68b6ab14e98b2734f289168550c3ec82ed46634ca110...*

*00dbe078160deb667451a2de203c92ec8e1e33e2e691a6cbdbaeb68b53506f44*

This hash **and** *JHk6hk* will be stored, associated with a given user

# Salt

Using salt makes it dramatically more difficult for attacker to recover passwords using dictionary-based attacks

- Also, even if users have the same password, they won't have the same hash (as their salted passwords will differ)

Back