Bitcoin Cryptography:
Hash Functions and Elliptic Curves

Ryan X. Charles
Blockchain University
San Francisco, Oct. 26 – Oct. 30, 2015

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Bitcoin Cryptography: Hash Functions and Elliptic Curves

Ryan X. Charles

Blockchain University

San Francisco, Oct. 26 – Oct. 30, 2015

Ryan X. Charles

Founder of Datt (datt.co)

twitter.com/ryanxcharles

github.com/ryanxcharles

Code Samples and Slides at:

github.com/ryanxcharles/blockchain-university

Datt

Outline

Hash functions

Base 58

Blocks, Transactions, Addresses

Elliptic Curves

ECDSA: Private keys, Public keys, Signatures

Bonus: Encryption

Cryptographic Hash Functions

- data → hash(data)
- Hash value is always the same number of bytes (e.g. 20, or 32)
- Can “easily” compute hash from data – difficulty is O(N)
- Computationally impractical to compute data from its hash
- Changing the data, even slightly, changes the hash
- Impractical to find two pieces of data with the same hash
- Hash value is “random” - all bits are equally possible

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Cryptographic Hash Functions

- md5 ← thoroughly broken
- sha1 ← known to be weak
- sha2 (sha256, sha384, sha512) ← sha256 used by bitcoin
- ripemd160 ← used by bitcoin

Cryptographic Hash Functions

- Data: “example data to be hashed”
- **sha1**: `c9e38c7dd778cdeffcdff7188c1a4c4d3c767e3b`
- **sha256**: `aa2356b0d9098e8fc3cdfe63a07a83c85533d06ea6270abf50e5ef7dc60620f9`
- **sha512**: `760e5df6338ca1a3218fb9f7e5a2e9d20984c3c707f918feb1231c8715636100ff2d484972db370e46b4f8eaaf3294a95191ad6e9f7ebef2d4ffb5c7bef3a13a`
- **sha256sha256**: `08dbfcea88a9fc1b9a91ca40de59bb0f30e99285ff4f4019f6b4c9509043b00b`
- **ripemd160**: `64988b52845db1905cbf2530a9f60c666c8836f2`
- **sha256ripemd160**: `200fbb165d5d9e31ad72ca7fa31618f5dfd4bfd7`

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Transactions and Blocks

- Transaction ID: sha256sha256 hash of the transaction – then always reversed when displayed
- Block ID: sha256sha256 of the block – then always reversed when displayed
- Hash of genesis block:
  `6fe28c0ab6f1b372c1a6a246ae63f74f931e8365e15a089c68d6190000000000`
- ID of genesis block:
  `000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f`

Base 58

- Base 10: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Base 2 (binary): 0, 1
- Base 16 (hex): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
- Base 64: 0 – 9, a – z, A – Z, and: ' ( ) , - . / : ?
- Base 58: similar to base 64, but only alphanumeric, and no confusingly similar characters like uppercase i and lowercase L. Set: 123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Base 58 Check

- What if someone is copying a Base 58 value by hand? Would be nice to have a checksum to check for errors ...
- Data + hash(data) → Base58
- Base58Check Encode: Find the first four bytes of sha256sha256(data), and append to data, then Base58 encode
- Base58Check Decode: Base58 decode, then parse data as everything but the last four bytes, and confirm that the last four bytes equal the first four bytes of sha256sha256(data)

Addresses

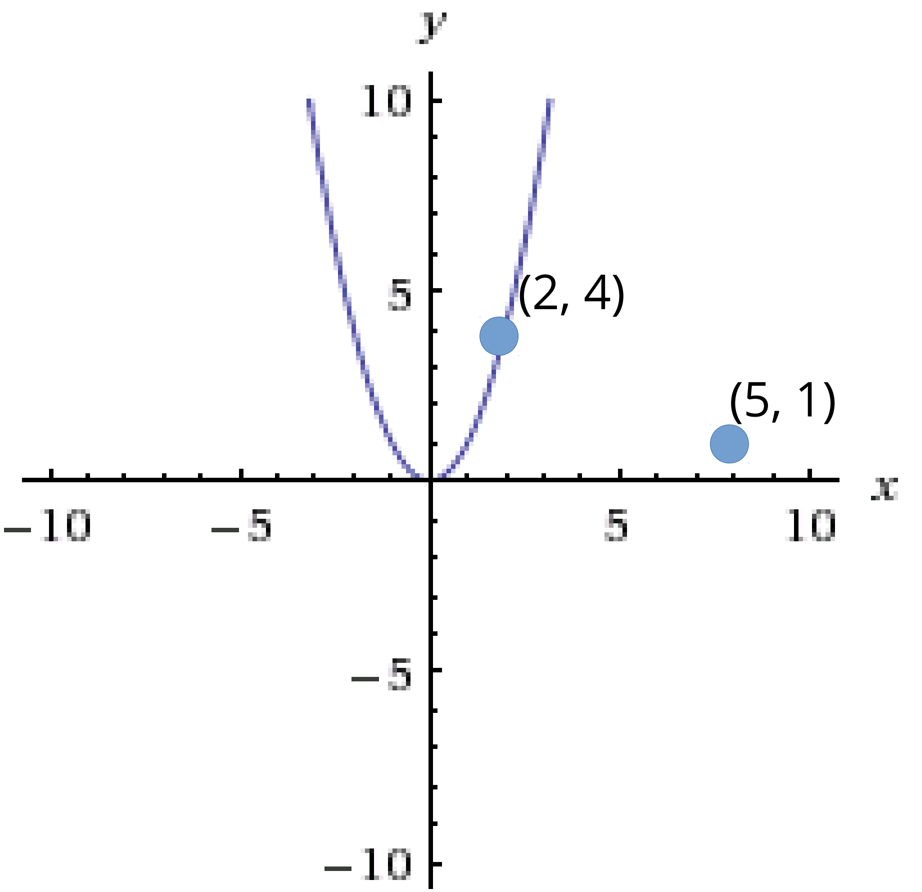
- An address is the sha256ripemd160 hash of a public key (usually a compressed public key)
- When displayed visually, addresses are displayed in base58check notation. They are also prepended with a 0x00 byte (if they are pubkeyhash addresses), or with a 0x05 (if they are a p2sh address)
- It so happens that when encoded this way, normal addresses always start with a “1” and p2sh addresses always start with a “3”
- Example address: 1343UaYEDYpbHQH2KF24mxTJ9GGXbzrGX9
- Example p2sh: 3GENUmnERtWfQct4NegomUqqtZuzYGPwBS

Some Context on ECDSA

- Bitcoin transactions involve “signing”
  - Technically, transactions don't need to involve signatures, but they almost always do
- ECDSA: Elliptic Curve Digital Signature Algorithm
  - ECDSA can: create a signature from data and privkey
  - ECDSA can: validate a signature from data and pubkey
- Signatures and public keys occur in transactions (private keys are kept private, of course)
- Bitcoin's core protocol does not use encryption
  - ECDSA is “cryptography”, but not “encryption”
- **ECDSA** is based on mathematics of **Elliptic Curves**

Basic Math: Numbers, Points, Eqns

- A number: 5
- A point: (5, 1)
- An equation: $y = x^2$
- A solution:
  - $y = 4, x = 2$
  - Or, (2, 4)



Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

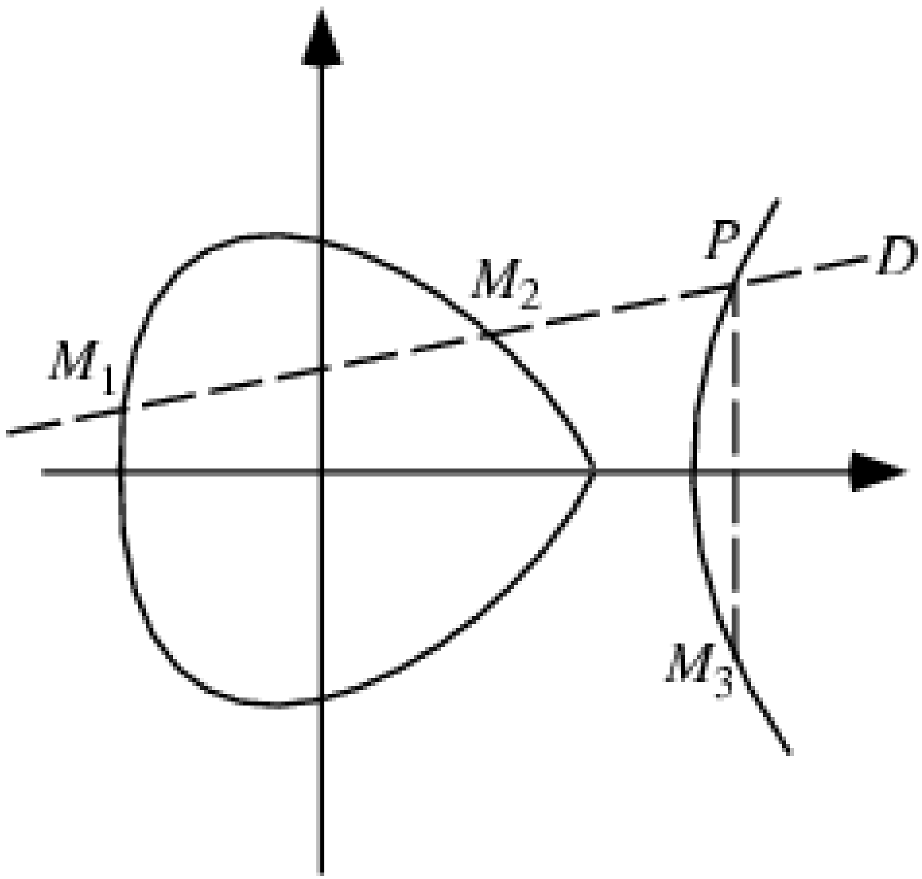
Elliptic Curves

Points on an EC are solutions to this eqn:

$$y^2 = x^3 + ax + b$$

$$M_3 = M_1 + M_2$$

Image Credit: Wolfram



Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Elliptic Curves

- Can define “addition” and “multiplication” of points to get a **field**
- A field is a set of points: $X = (x, y)$
  - With addition operation: $(x_1, y_1) + (x_2, y_2) \rightarrow (x_3, y_3)$
  - Multiplication defined as: $2 * (x, y) \rightarrow (x, y) + (x, y)$
  - Commutative: $(x_1, y_1) + (x_2, y_2) = (x_2, y_2) + (x_1, y_1)$
  - Associative: $\mathbf{(}(x_1, y_1) + (x_2, y_2)\mathbf{)} + (x_3, y_3) = (x_1, y_1) + \mathbf{(}(x_2, y_2) + (x_3, y_3)\mathbf{)}$

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Elliptic Curve Cryptography

- Cryptography you can do with elliptic curves:
  - Sign and verify data
  - Encrypt and decrypt data
- EC Crypto is based on **finite fields** – not continuous curves as you might think
  - The same equations and operations apply – but cannot be visualized as a curve
- Confusingly, finite fields do not look like curves

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Elliptic Curve Cryptography

- Mathematically possible to subtract:
- If A + B = C, then B = C – A
- …but this is computationally impracticable: equivalent to solving Discrete Logarithm Problem
- A = 2 B = B + B …
- If you know “A” and “2”, you can only find B from brute force – try every value until you get it.
- Quantum computers can solve discrete log problem – problem for EC crypto!

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

secp256k1

- The curve used by bitcoin – specified by curve and finite field parameters: (a, b, p, n, G, h)
- $y^2 = x^3 + ax + b$ (as always for elliptic curves)
- $a = 0, b = 7$
- $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 – 1$
- n = 115792089237316195423570985008687907852837564279074904382605163141518161494337
- G = (55066263022277343669578718895168534326250603453777594175500187360389116729240, 32670510020758816978083085130507043184471273380659243275938904335757337482424)
- h = 1
- N: “order of the curve” = “largest private key”
- G: “base point of the curve”

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Private Keys and Public Keys

- A private key is a number in the range (0, N)
- A public key is a private key times G
- Public Key = Private Key * Base Point
- P = pG
- If p = 5, then P = 5G, or:
  - P = 5 * (x, y)
    - P = (21505829891763648114329055987619236494102133314575206970830385799158076338148, 98003708678762621233683240503080860129026887322874138805529884920309963580118)
- Never use “5” as a private key! Generate a random number in range (0, N)

Private Keys and Public Keys

- A randomly generated private key and corresponding public key
- p = 79166932645790398746161737314012976057631666974820812284863365219350761232064
- Bitcoin base 58 check format: L35wVgKWPyfeMzNUWQR3D2MtjCUzMyrvFpN9R44NTZbhcCea6kyS
- P = (14940432162225245106778043682370687268864377313694688834137709913105567990260, 22312549033088701518791571940340055302090435747572883935510928095745029494859)
- DER compressed hex format: 032107fc24b3569f2feda301c7efc07f56fc8ab6870e61f03dcd93dbd9088a41f4

```
fullnode> var privkey = Privkey().fromRandom()
fullnode> privkey.bn.toString(10)
'79166932645790398746161737314012976057631666974820812284863365219350761232064'
fullnode> privkey.toString()
'L35wVgKWPyfeMzNUWQR3D2MtjCUzMyrvFpN9R44NTZbhcCea6kyS'
fullnode> privkey.toString()
fullnode> var pubkey = Pubkey().fromPrivkey(privkey)
fullnode> pubkey.point.getX().toString(10)
'14940432162225245106778043682370687268864377313694688834137709913105567990260'
fullnode> pubkey.point.getY().toString(10)
'22312549033088701518791571940340055302090435747572883935510928095745029494859'
fullnode> pubkey.toHex()
'032107fc24b3569f2feda301c7efc07f56fc8ab6870e61f03dcd93dbd9088a41f4'
```

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

ECDSA

- Given private key and data, need to produce a signature
- Given signature, data, and public key, need to confirm that signature is for corresponding private key
- ECDSA: Elliptic Curve Digital Signature Algorithm
- An ECDSA signature consists of two numbers, r and s. Although a pair of numbers, it is not a point.

ECDSA - Sign

1. Calculate $e = \mathrm{HASH}(m)$, where HASH is a cryptographic hash function, such as SHA-2.
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n.
3. Select a **cryptographically secure random** integer k from $[1, n-1]$.
4. Calculate the curve point $(x_1, y_1) = k \times G$.
5. Calculate $r = x_1 \bmod n$. If $r = 0$, go back to step 3.
6. Calculate $s = k^{-1}(z + rd_A) \bmod n$. If $s = 0$, go back to step 3.
7. The signature is the pair (r, s).

- Pseudocode credit: Wikipedia
- (Caveat: Not all details here – do not rely on for implementation)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

ECDSA - Verify

1. Check that Q_A is not equal to the identity element O, and its coordinates are otherwise valid
2. Check that Q_A lies on the curve
3. Check that $n \times Q_A = O$

After that, Bob follows these steps:

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \textrm{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $w = s^{-1} \bmod n$.
5. Calculate $u_1 = zw \bmod n$ and $u_2 = rw \bmod n$.
6. Calculate the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$.
7. The signature is valid if $r \equiv x_1 \pmod{n}$, invalid otherwise.

- Pseudocode credit: Wikipedia
- (Caveat: Not all details here – do not rely on for implementation)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

ECDSA – Extra Details

- It is possible to derive candidate public keys from a signature – which can be compared to the hash of a public key to see if a signature is valid. Makes possible “**Bitcoin Signed Message**”
- Generating a signature involves a random value “k”. This value needs to be unpredictable, but can be computed deterministically from the data and the private key – standard for this is **RFC 6979**. This is actually safer than a purely random value, in case your entropy pool is broken. Also helps validate the signature algorithm is working correctly by comparing to known test vectors.

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

ECDSA - Summary

- ECDSA Sign: Privkey + hash(data) → (r, s)
- ECDSA Verify: (r, s) + hash(data) + Pubkey → valid|invalid
- Computationally impractical to derive privkey from pubkey
- Computationally impractical to derive privkey from signature
- Use RFC 6979 – deterministic k

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt