Dev++ 2017 Notes

Prereqs: follow the instructions in the following two repos

- https://github.com/BitcoinEdge/devplusplus
- https://github.com/BitcoinEdge/bitcoinrpcapp
- Also install Bitcoin Core or for mac can use tar here
    - Manually launch from command line w/ testnet flag
        - `./src/bitcoind -testnet`
    - Use the following debug.log file to monitor
        - `tail -f $HOME/Library/Application\ Support/Bitcoin/debug.log`

Foundational Math / ECDSA / Transactions
(See Dev++ - Jimmy Song in presentations & follow instructors for when to use each Jupyter notebook)

- Finite Field
    - Set of numbers
    - Closed under +, -, *, / except division by 0
    - Used with Elliptic curves for cryptography
    - Prime fields - prime number of elements in the field
        - Eg. $F_{19} = \{0, 1, 2, .. 18\}$
    - Addition & Subtraction defined as modulo arithmetic
        - 11 + 6 = 17 % 19 = 17
        - 17 - 6 = 11 % 9 = 11
        - 8 + 14 = 22 % 19 = 3
        - 4 - 12 = -8 % 19 = 11
    - Multiplication and Exponentiation also defined as modulo arithmetic
        - 2 * 4 = 8 % 19 = 8
        - 7 * 3 = 21 % 19 = 2
        - 11^3 = 1331 % 19 = 1
            - 1/n = `pow(11, 3, 19) == 1`
    - Division - inverse of multiplication
        - 2 * 4 = 8 --> 8 / 4 = 2
        - 7 * 3 = 2 —> 2 / 3 = 7
        - 15 * 4 = 3 —> 3 / 4 = 15
        - Most expensive operation
        - Fermat's Little Theorem - only for prime fields
            - n^{p-1} = 1 mod p
            - n^{p-1} = 1 —> 1/n = n^{p-2}
                - P is the prime of the field eg. 17 in example below
                - n = 15 in example below
            - 3 / 15 = 3 * (1/15) = 3 * 15^{17} = 4
- Elliptic Curves
    - $y^2 = x^3 + bx + c$
    - Bitcoin curve, secp256k1: $y^2 = x^3 + 7$
- Point Addition

    - If a line intersects an elliptic curve at two points, it will intersect a third time
        - Exception: if points are exactly opposite each other
    - Group law for point at infinity - point at infinity acts like 0
        - $(x_1, y_1) = (x_1, y_1) + (\text{infinity}, \text{infinity})$
        - Find 3rd point:
            - $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$
            - s = (y2 - y1) / (x1 - x2)
            - x3 = s^2 - x1 - x2
            - y3 = s(x1 - x3) - y1
    - Addition is highly non linear - property to take advantage of
- Elliptic Curves over Finite Fields
    - Elliptic curve over finite field looks scattered but equations still hold true
- Mathematical Group
    - Single operation - closed, associative, commutative, invertible, has identity
    - Point addition results in a group
- Scalar Multiplication
    - Start w/ an Elliptic Curve over finite field, pick a point G, a generator point, keep adding until get to point at infinity - creates a finite group
    - Imagine very large group $n \sim 2^{256}$
    - P = sG where s is really large
        - Easy to find P when know s, but finding s when we know P is difficult
        - Secrete Exponent
        - Discrete Log Problem
            - $P = G^s \longrightarrow \text{Log}_G P = s$
- Public Key Cryptography
    - Private key is the scalar (s) - one 256 bit number
    - Public key is resulting point sG (P)
        - G is generator point
        - Point (x,y) thus two 256 bit numbers
- SEC Format - Standards for Efficient Cryptography
    - Public key (point on curve) serialized
        - Uncompressed - 04 as marker, x coordinate 32 bytes, y coordinate 32 bytes —> 65 bytes
        - Compressed - 02 if y is even, 03 if y is odd as marker, x coordinate 32 bytes —> 33 bytes
- Addresses
    - Take either compressed or uncompressed SEC format
    - SHA-256 the result & then RIPEMD the result (HASH160)
    - Prepend with network prefix (00 main net, 6F testnet)
    - Add 32-bit double-SHA256 checksum at the end
    - Encode in Base58
- ECDSA (Elliptic Curve Digital Signature Algorithm)
    - sG = P
        - s = secret, p = public key, g = generator point
    - uG + vP where u != 0, v != 0
        - Choose u & v, can only manipulate sum if you know how G & P are related (know private key), otherwise random point on serve
    - uG + vP = uG + vsG = (u + sv)G
        - If you know s, you can manipulate u + sv
- Signature Algorithm

- Start with hash of what you're signing (z)
    - Assume secret is e and public point P = eG
    - Random number k
    - Compute kG, x coord = r
    - Compute s = (z + re) / k `pow(k, n-2, n)`
        - s = (hash + x coord(secret)) / random number
    - Signer can compute s since they have e
    - Signature is (r, s)
- Verification Algorithm
    - Start with hash of what you're signing (z)
    - Assume you have public point P = eG
    - Signature is (r, s) where s = (z + re) /k
    - Compute u = z / s
    - Compute v = r / s
    - Compute uG + vP = (z/s)G + (r/s)P…= (r,y)
        - If x coord matches r then sig is valid
- Transaction
    - Assignment of bitcoin from one script to another
    - Address - compressed script
    - Version (4 bytes), inputs, outputs, locktime
        - Inputs - coinbase, previous transaction output(tx out, utxo, spendable, outpoint)
    - Input - previous tx hash, Vout(index in that txn), sequence (RBF), Scriptsig - all nodes must validate
        - RBF - replace by free, replacing one version of an unconfirmed txn with a different version that pays a higher txn fee
    - Output - Amount, ScriptPubKey
    - Locktime - tell nodes not to let txn into block until certain block height
- Script
    - Programmable way to assign BTC
    - Two items: elements & operations
        - Elements - data (signatures, keys, hash)
        - Operations do something to elements
            - eg. OP-DUP, OP_HASH160, OP_CHECKSIG, OP_RETURN
        - Added to stack
    - At the end of processing all items, there must be a single non zero elements on the stack to evaluate to True for txn to be valid
    - Add elements to stack & use op to process
    - P2PKH - more secure that P2PK
- Transaction Validation
    - Check unspent, inputs >= outputs (diff is tx free to miner), check scriptSigs are valid (sub scriptSig with empty, sub with pubKey, append hash type, double sha256 to get hash (z), get signature (r,s) and pubKey from scriptSig - use to verify input
    - Quadratic hashing - segwit fixes
- Pay to Script Hash
    - Addresses start with 3
    - Flexible addresses because part of Script, RedeemScript kept by creator and must be provided when spending - treated as element & interpreted as script later
    - Post BIP-16, RedeemScript only added to queue if hash matches last hash in ScriptPubKey
    - To verify, replace scriptSig with 0 & put in RedeemScript & add Sighash, and hash all

of that and then all of that gets signed

Blocks / Blockchain / Network
(See Blocks & BlocksPresentation in presentations & follow instructors for when to use each Jupyter notebook)
prezi

- Difficulty - human friendly way of expressing target
- Can soft fork difficulty increase but not difficulty decrease
- Block structure - block header (information about the block 80 bytes), txns (up to 4mb post sec wit, initially 1mb)
- Block header - block version, hash of previous block (output of sha256 hash), nonce, timestamp, merkle root , difficulty bits
- Coinbase txn = always the first txn in the block, pays out coinable reward to miner (block subsidy + txn fees), contains block height, Merkel root of transaction witnesses, extra nonce space
    - Only has one input, outpoint is null (in normal txn it points to UTXO it is spending), prev txn is 0 and index is -1
    - ScriptSig - starts with block height BIP34 & can be up to 100 bytes
    - Can have many outputs - txn outputs add up to block reward + sum of all txn fees in block
    - Since segWit one of the outputs must commit to the witness root - how SegWit is a soft fork
    - Can't be spent for 100 blocks after mined - helps control for reorgs
- Timestamp set by miner - for block to be valid timestamp must be after the median time of the previous 11 and less than two hours beyond the current time system
- Txns are hashed into 32 byte digest
- Merkle Tree - permits compact proof of inclusion to show that txn included within block w/out revealing contents of set
    - Path of Merkle tree through the txn
    - Cannot show proof of exclusion
- SPV wallets (Simple Payment Verification) - make use of Merkle proofs
    - Ask full node for proof that txn is in the blockchain
    - Prover provides headers chain and merkle proof to show that txn exists
        - No proof of exclusion thus prover can 'lie by omission' and not reveal presence of txn
    - SPV should query multiple full nodes to insure reliability
- Forks & Re-orgs
    - Majority decision represented by longest chain which has created PoW - assumption that longest chain has greatest PoW, issue in original implementation
    - Valid chain - most accumulated work
        - 2 chains of same accumulated work - tie break is which chain was seen first
    - Temporary chain fork - expected
        - Block discovery is random - if two blocks discovered at same time then chain split since different notes will see diff ones first
        - Chain forks resolved when next block found on one chain and becomes one with most PoW - reorg
- SegWit - fixes txn malleability (same txn but can change signature (diff random number, diff signature) thus hashes to diff value), fixes Quadratic hashing, increased block size
    - Adds witness to txn serialization - more space in block for txns

    - Witness commitment in header - coinable txn must commit
        - OP_RETURN followed by some bytes followed by merkle root of witness in block - places in scriptPubKey of one of the coinbase txn outputs
    - With SegWit, signature moved to witness thus no longer included in txid thus not hashing it with inputs thus removing malleation
- Soft fork - constrict what's possible, Hard fork - expand what's possible
    - Soft fork can be done in forward compatible manner since new rules still valid under old set of rules
- Peer-to-peer network
    - How txns and blocks are propagated
    - Nodes initially connect o seed nodes - gossip via ADDR
    - Bitcoin Core will connect to up to 8 outbound peers
        - Nodes may or may not accept inbound peers
    - Nodes that misbehave need to be removed
        - Bad behavior - invalid txns or blocks, unconnected blocks, stalling, non standard txns, malformed messages
    - Full node - verifies validity o blocks and that all txns are valid - enforces consensus rules of bitcoin network
    - Pruned node - type of full node but discards old block data to save disk space - retains at least 2 days of blocks and undo data to allow for re-orgs
        - Propagates new blocks but cannot serve old blocks
    - Archival node - rating all old block and undo data and can serve old blocks to pees on network - NODE_NETWORK
    - SPV Node - only downloads block headers and merkle proof of specific txns - can validate PoW but can't validate other network rules, invalid or double spend txns, or money supply
        - Bloom filters - preserve some privacy, doesn't exactly tell full node what addresses - get what needed from full node but they can't see exactly what you want
    - blocksonly - full node that doesn't propagate txns outside of blocks
    - nolisten - node which makes outbound connections but doesn't accept inbound
    - onion - connect to peers using tor
    - proxy - connect to peers via proxy
    - whitelist
    - Message format - p2p messages have header & payload
        - Magic (4 bytes) - indicated network
    - Control messages
        - p2p connection starts w/ version handshake
        - used by nodes to exchange information about themselves
        - Version (4 bytes) - highest version transmitting node can connect to
        - Services (8 bytes) - bitfield of services supported by transmitting node
        - Timestamp (8 bytes)
        - adds_recv, nonce, user_agent, start_hieght, relay
        - VERACK - sent in response to VERSION message
        - ADDR - gossips
    - Headers-first syncing
    - Compact blocks - reduce time & bandwidth for propagating blocks
        - Low bandwidth - same # of message as headers first block syncing but saves on # of txns sent
        - High bandwidth - quickly propagate txns around network, saves on propagation time

        - Doesn’t included all raw txns in a block
        - Excludes txns that transmitting node thinks receiving node has already seen in mempool
            - Receiving node reconstructs block using txns in mempool & can use GETBLOCKTXN to fill gaps
    - Inventory / data messages
- Mempool
    - Node’s view of unconfirmed txns
        - txns propagated around the network using INVs are stored in nodes’ mempools
        - miners select ins for the next block from their mempool
        - txn chains are allowed - can spend from uncofirmed txns
    - Each node’s mempool is a private resource & we have to limit how much it can be used
        - Expire out old txns - txn there longer than 14 days then age it out
        - Limits mempool size - 300MB - when full, evict low fee paying txns
            - Want to keep txns that will eventually get into block
            - Fee-rate calculation done by ‘package’ - chain of unconfirmed txns - miner will include ancestor with low fee and child with high fee because need parent for inheritance and want high txn fee
            - Node can tell peers it no longer wishes to receive txns below certain fee
    - Replace by fee
        - Unstick txn - replace old txn w/ new version with higher fee
        - Miner can choose to include any version of txn
        - OPT-IN (BIP 125) - allows users to signal that replaced later is allowed - use sequence number in txn # to opt in
        - Could be a DoS vector - send txn and continuously bump up fee
            - Reqs: One input from original txn has sequence #, higher fee, no new unconfirmed inputs, pays for own bandwidth, does not replace more than 100 txns
    - Transaction Caching
        - Seen twice - once when broadcast by user and ˙accepts to mempool and then second when included in block
            - Cache first time so don’t need to validate on second seen
        - Signature Caching
            - Most expensive part of txn validation - each txn input usually includes at least one ECDSA signature - keep a cache of signature validations
            - Edge case DoS protection
        - Script Caching
            - Caches txn and validation of entire script
            - Sciprt validity does not depend on any data outside txn
        - blocksonly node won’t benefit
        - Front loading block verification
        - Can’t cache validity of entire transaction because they are contextual
    - Fee Estimation
        - Txn fee - implicit
        - Fee rate depends on demand for blockchain space
        - Use mempool - gives idea of current competition but expected time to wait for block is 10 min so doesn’t account for what will arrive in next 10 min & doesn’t account for luck / unluck
        - Look at recent block - trivially gamble by miners who could stuff block with private, high-fee paying txns to artificially make price lok high

        - Bitcoin Core considers historic data in mempool & recent blocks - requires nodes to be running for some time with a mempool

HD Mining
(See HDWalletsPresentation in presentations)

- BIP 32 - HD Wallets
    - Prevent address reuse - construct a tree of keys
    - Allows subbranches to be shared individually - linear would mean that once one person has one private key, they can generate all the rest
    - Generate random 128-512 bit seed
    - Use HMAC (Hash Message Authentication Code) to derive child nodes
- BIP 39 - Mnemonics for HD wallet seeds
- BIPs 43 & 44 - multi wallet hierarchy
- Chain analysis - can determine who owns specific bitcoin - bad for privacy
- Reusing addresses - bad for privacy
- ECDSA requires cryptographically secure ephemeral key - signing with same ephemeral key reveals private key
    - If PRNG (pseudo random number generator) is broken, reusing address can reveal private key
- Sending an address using P2PKH does not reveal public key
    - ECDSA is not quantum secure
- Recurrent txns - give public key & chain code - payee sending multiple txns to diff addresses you own
    - Non hardened key

Mining
(See BitcoinMining in presentations)

- Strartum mining protocol - created to replace network protocol
- Mining subscribe - response with difficulty
- Pool Attack Vectors
    - Block withholding - very difficult to detect & costly - can not be fully mitigated without hard fork
        - Can potentially reduce difficulty of bitcoin network
    - DDOS - not as common currently
- Stratum Attack Vectors
    - BGP (Border Gateway Protocol) Hijacking
        - Can partition network & delay block delivery
    - DNS Cache poisining
    - Cloud host interception

Hostile Actors & Attack Vectors
(See Hostile Actors & Attack Vectors in presentations)

- Confirmations
    - Confirmation count of txn goes up as more blocks on top of block in which that txn is contained

- Reorgs
    - Blocks found simultaneously - chain which gets more PoW quicker becomes the legitimate one
- Txns compete w/ each other if they use the same inputs
- Segwit solves double spending through malleability

Developing Apps on top of Bitcoin Core
(See DevPlusPlus-RPC_App in presentations)

- Don't encrypt your wallet for now, otherwise have to put in password every time - can encrypt at end when you are done
- getbestblockhash - gets most recent block
- Emphasis: don't use an address more than once

Lightning Network
(See devpp lit in presentations)

- "double spends" - only one txn can get confirmed into block
- Network of channels allows for one-off payments —> multi hop —> assume all can talk, but don't have payment channels open
    - Could use small payments eg. 1/100th of coin to minimize risk
    - OR - Dave comes up with random number, takes hash of it, and sends the hash to Alice
        - Alice pays Bob a coin —> 3 txn outputs, spendable by Bob if Bob knows presage of H (Bob needs to sign and post R, or another preimage) OR Alice can take the coin if Bob doesn't sign + R within a day —> Bob doesn't have R thereby infeasible for Bob to find R, so listens to Alice and forwards to Carol
            - Nodes only know of nodes next to them thus Alice doesn't know about Carol
        - Carol creates same output spending to Dave & Dave can redeem if knows R, which he does
            - Dave can close channel & broadcast txn by revealing R —> if Carol goes offline Dave would have to do this
                - If Carol can't get ahold of Dave she can go back to Bob & cancel txn - make new state in channel back to before this state
                - If Bob & Dave are offline, Carol is just holding on to coin for which she doesn't know R, if she can find R, she can broadcast & take the money, or she can close channel by broadcasting the state & doesn't get the money
            - Instead, Dave just tells Carol what R is, Carol tells Bob what R is, Bob tells Alice what R is, and then Alice knows that payment was successful
                - R is stored in signature script
            - Thus update channel state to not update R since everyone in channel is aware
            - These are HTLCs (hash time locked contracts)
        - Bob —> Carol same way of Alice —> Bob
    - Lightning fees ??
    - Backups are bad - could accidentally broadcast wrong state & lose all $$ - if you forget, don't do anything and wait for counterparty to close out channel eventually -

    backups ok if have multiple computers with real time backups
    - Fees to miners set by Fund txout —> Bitcoin miner only sees last txn - don't care about all the lighting state changes
- Nothing at Stake for Alice is she has nothing in channel - incentive to prematurely close
- Multi hop over different channels - doesn't matter what blockchain those channels are on, Alice could have Bitcoin channels with someone & that person could have altcoin channels tie Carol
- Only want most recent state on chain - how to tell which state is the right one? all look the same to the network
    - Have different script for which is Alice waits 100 blocks to be able to spend txn, but if she tells Bob the secret preimage, he can take the $$ - if Bob knows the secret he can take all the BTC, but if Alice broadcasts a new block, then Bob no longer knows secret and must redeem w/in 100 blocks otherwise it's Alice's
        - Homomorphically add Bob's signature & Bob's (PubKey + another PubKey) homomorphically added & Alice has private key for this — ??
    - What happens when broadcast old state - counterparty takes all the money — counterparty broadcasts justice txn
    - Penalty txns in white paper == justice txns in code
        - Both parties must have $$ in the channel for justice txns to work - required minimum amount on each side now
- Lit message flow - fund
    - Channels are all funded by single party right now in current implementation
    - Alice connects to Bob gives her PubKey and asks for his PubKey
    - Alice then connects back and provides description of payment channel, Bob can then acknowledge channel, which contains first transaction, Bob gives signature sending to first state txn
    - Alice so far has only revealed TXID, Alice hasn't proved she has any $$ but Bob has no risk
    - Alice funds channel by broadcasting txn - shows proof its in block & then they have a channel
        - Bob just looks for txn on blockchain, no proof implemented yet
- Lit message flow - push
    - Alice & Bob have diff amounts in their channel, at the same state
    - Alice sends 3 coins to Bob, she is losing -3 during state update, but prior to send, still have 66
    - Alice creates message, updating state, sends 3 coins, & sends signature
        - Bob updates state to updates amount to 28, and updates delta to 3, thus Bob has more $$ and can close since he received Alice's signature, Alice can't close yet until Bob provides signature & revocation (secret R, so he can't broadcast 4 without losing money)
            - Bob & Alice both have different secrets for each state
            - Bob can broadcast 4 or 5 before he sends Alice his state 4 secret, when he sends Alice his state 4 secret, he cannot broadcast state 4 - this whole time, Alice can only broadcast state 4 since she hasn't told Bob her state 4 secret event though they are both in state 5
        - Reverse Merkle tree for secrets (Elkrem)
            - If you know the root you can get all the leaves
- Can close channels by sending to multiple addresses ??
- Play around with this: https://github.com/mit-dci/lit

Discrete Log Contracts
(See DLC devpp in presentations)

- Invisible bitcoin smart contracts
- Private keys a & b
    - aG = A
    - bG = B
    - C = A + B = (a + b)G —> if Bob gives b to Alice she can sign with C
- Conditional payments - payment contingent on external data
    - Need some sort of oracle to reference external state with something like Lightning
- 2 of 3 multisig oracle - interactive meaning they see & decide the outcome of every contract - would be better if they couldn't equivocate & never saw contracts
- Fix the R point of the Schnorr signature - thus can only sign once
- For any given message, can computer sG —> signature time generator - Discrete Log Problem
    - Use sG as public key, s as private key (Olivia's signature)
- DLC - dont' need to watch for fraud like necessary in lightning
- Oracle cannot deviate in answer between contracts eg. all rain or all sun can't tell some sun or some rain without revealing private key
    - Oracle signs publicly - no motivator for specific oracle, need reliable data feed
        - Examples like Bloomberg … - not way to specifically secure endpoints different from trusting already logged business sources
        - Could aggregate oracles ?
- Whole process is 3 txs: fund, close, sweep
    - Sweep by broadcasting another txn
    - If trust, then 2 txns: fund, gg
    - GG txn - if everyone agrees, create a new txn at closing time which sends to unencumbered outputs
        - If they cooperate they can update channel abalone to reflect difference in contract execution - nothing touches blockchain yet, open state channels
- Novation - if Alice & Bob online & agree, can create GG txn, but is interactive between computers & humans thus is unlikely - Alice will want to get out with $$ and Bob will want to mitigate losses
    - Better —> Alice replaces herself with Carol —> interactive from computer sense but not human sense
    - All 3 need to sign, but Bob can be automated

Atomic Swaps
(See Dev++ - Crosschain swaps)

- Trade either happens or doesn't happen - never the case that one gets the coins the other does not
- Useful to trade across blockchains & for privacy to obfuscate transaction graph (mix coins)
- W/in same blockchain, if Alice creates tx3, then tx3 needs Alice & Bob's signatures to spend Tx1 & Tx2 - Tx3 either on blockchain or not
- Non Atomic Cross Blockchain Trades - Bob is able to cheat Alice
    - Prevent w/ hash locks
- Hash locks - spending condition, to spend a txn output, you must provide val x such that h(x) = y
    - Bob needs signature + x to be able to spend txn

- Atomic swap - Alice can only claim coins by proving X which allows Bob to claim coins
    - Add time lock to prevent unspendable funds
- https://github.com/NicolasDorier/XSwap