# Compact Confidential Transactions

Denis Lukianov

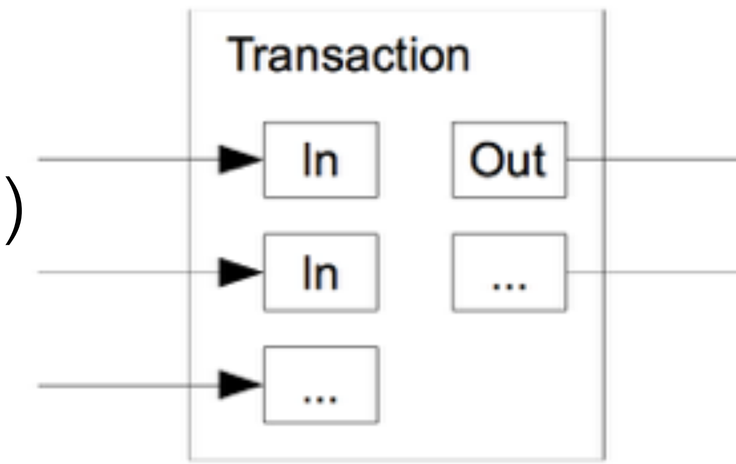
16 December 2015

Disclaimer: Challenge and verify!
Warning: Large curves!

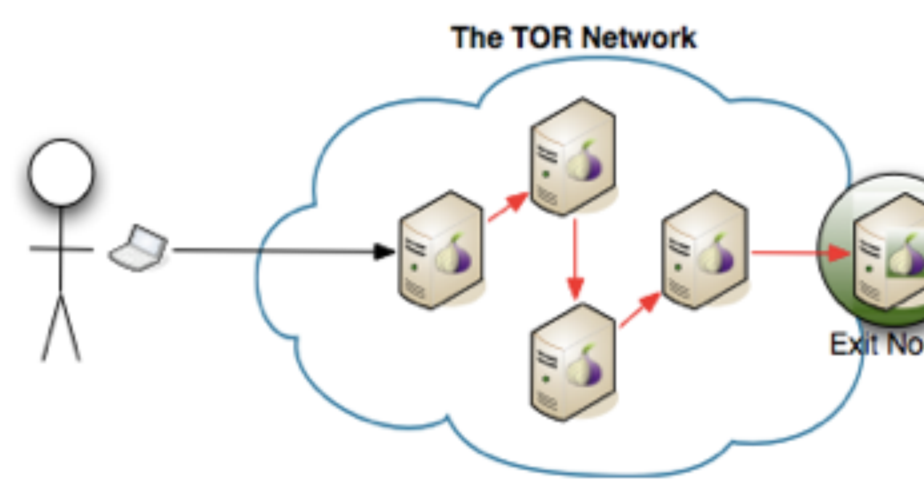
Paper: http://www.voxelsoft.com/dev/cct.pdf

# Transaction privacy

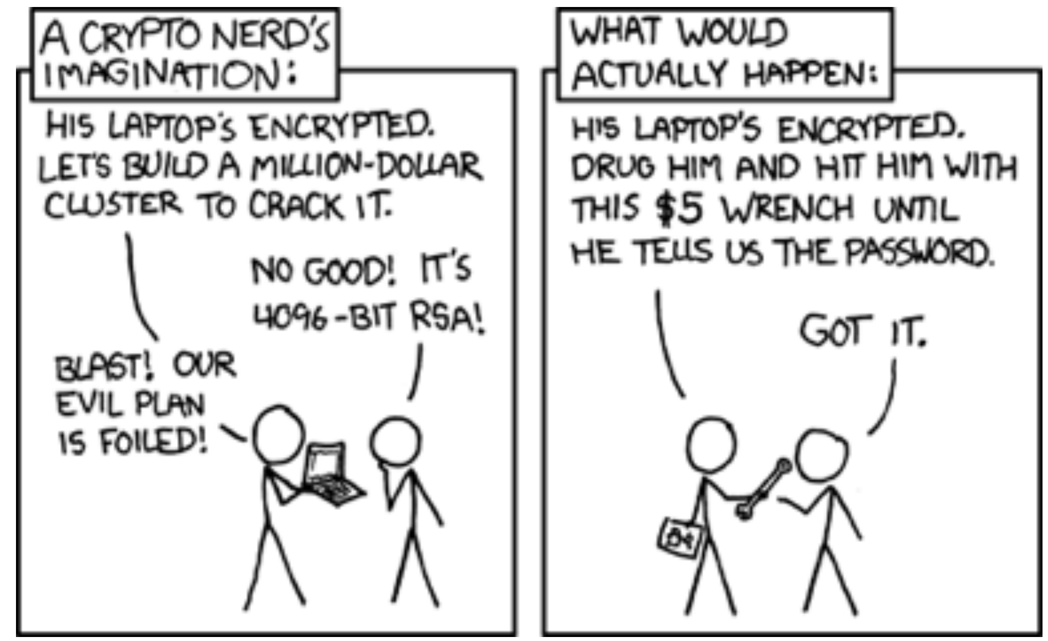
- Unlinkability
  - Find where an output went? (ZC, Stealth Address)




- Untraceability
  - Find where the output came from? (ZC, Cryptonote)
- **Confidentiality**
  - **Find the value of the output? (ZC, CT, CCT)**
- Origination
  - Find physical sender/receiver/originator?



# Why confidentiality?




- More like cash
- Improves fungibility
- Keep salaries, rents, secret business costs and politically/culturally sensitive spending private
- Regulatory proof that only white-listed entities transacted, without disclosing how much
- Hinders prioritisation of participants for cryptanalysis

# Confidential Transactions

- Only **sender** and **receiver** should know output value
- **Everyone** needs to know that Sum(inputs) = Sum(outputs)
- Can we do this with crypto?
  - Proposed by Dr. Adam Back in 2013 on bitcointalk
  - Need a space efficient proof (comparable to txn of 600 bytes)
  - CT Elements with “Borromean Ring Signatures” (2015)

# With Borromean Rings

- Gregory Maxwell, Andrew Poelstra
- Initial 2013 idea: commit to every bit, prove it 0 OR 1 (Pedersen)
- Then use ring signatures of multiply-chameleon hashes to combine the “OR”/“AND” proofs
- Advantages
  - Use existing curve, 1300 verifications/sec
  - Somewhat compact, 2.5kB (600 byte txn)
  - Works on useful integers

# Alternative approach

- ECC is deterministic, commutative, associative
- Cipher text equality guarantees plaintext equality
- Proof of sums in 0 bytes
  - $v*G = q*G + w*G$
  - $V = Q + W$
- But…

# Challenges

- Secrecy is vulnerable to brute force
  - Easy to get cipher-text of common values
  - Only $2^{52}$ combinations for the rest
- Integrity is vulnerable to modular overflow
  - Negative values can do this intentionally
  - Sum overflow allows the sender to mint coins

# Maintaining secrecy

- Add a large nonce in lower bits of 64-bit value
  - $x = value * 2^{fuzzbits} + U(0, 2^{fuzzbits})$
  - 220 bits to deal with giant-step baby-step algorithm (110 bits of added security)
  - 220 + 64 = 284 bits for our x
- We're gonna need a bigger curve!

# Maintaining integrity

- Only need to handle addition with small number of addends
- Each positive addition overflows by 1 bit
    - Allocate top 8 bits to allow 255 outputs
    - Must prove each addend is small enough
        - (we just made them bigger, but this is relative)
    - Must prove each addend is positive
        - (is there a cheap way to do this in zero knowledge)

# Interval proofs

- Chan, Frankel, Tsiounis (CFT)
  - “Easy Come - Easy Go Divisible Cash”, 1998
    - **Widened interval proof** in only 0.241kB
- Fabrice Boudot
  - “Efficient Proofs that a Committed Number Lies in an Interval”, 2000
    - **Square proof** also efficient (Discrete Log Equality)
    - Specific interval range proof [a, b] still quite expensive, 1.692kB
- Zhengjun Cao
  - “An Efficient Range-Bounded Commitment Scheme”, 2007
    - Adopting a single base

# Square proof

- Commit to E=x*G and F=x*E=x*x*G
- Pick random r, (Schnorr) commit to U = r*G, V = r*E
- Prove knowledge of multiplicand c = HASH(E|F|U|V) such that:
    - the multiplication and sum holds for U and V
    - the multiplicand cannot be pre-calculated (Fiat-Shamir)
- m = r + c*x (mod n)
- Verifier only needs (E, F, U, V, m) or, for space efficiency, (E, F, m, c)
- Since r = m - c*x, then U = m*G - c*x*G
- Verifier checks c = HASH(E|F|m*G - c*E | m*E - c*F)

# Widened interval proof

- Knowing x in [0, b], proving x in some much wider [-T, T], T = $b*2^{t+l}$
- Commit to E=x*G
- Pick r in [0, T], commit to R = r*G
- Prove knowledge of multiplicand c = HASH(E|R) such that:
    - the interval rules are met
    - the multiplicand cannot be pre-calculated
- m = r + c*x
- Verifier only needs (E, R, m) or, for space efficiency, (E, m, c)
- Verifier checks c*b < m < T and c = HASH(E|m*G-c*E)

# Security parameters

- t=128 is Schnorr parameter, number of bits in HASH
- l=20 is Zero-knowledge parameter from CFT
  - m = r + c*x
  - Sum of two uniform numbers is not uniform!
  - But it is uniform enough if $2^l$ is large
  - Makes statistical attack impractical
  - Infinitesimally Small Knowledge is Zero Knowledge
- fuzzbits=440 is the size of the nonce in lower bits of x

# Sum of squares per output

- Widened interval [-T, T] is not sufficient
  - Relies on RSA unfactorable group order
- Specific range proof [a, b] is expensive
- Can we use Boudot's square proof?
  - Warren Smith, "Cryptography meets voting", 2005
    - Every positive integer is sum of 4 squares
    - Every integer 4y+1 is sum of 3 squares
      - Zero knowledge proof for a sum of squares
      - Requires at least 6 ECC commitments

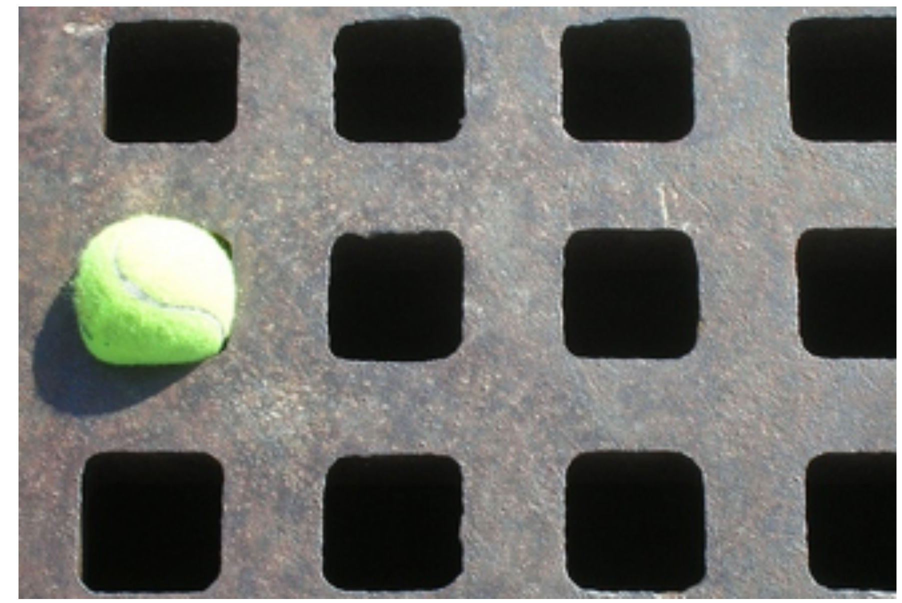
# Widened interval with known group order

- Widened interval relies on unknown group order, not valid for ECC
  - m = r + c*x
  - If prover picks a modular inverse, modulo group order
  - e.g. pick x = (N-1)/2
  - x is the encryption of “divide by -2” and verifier is fooled for even c
- But we can require another interval proof on (x+1)
  - Inverse moduli are unlikely to be adjacent

# If only…

- Maybe very efficient to combine proofs
    - CFT’s interval proof (E, m, c)
    - Boudot's square proof (E, F, m, c)
- If only every output value was already a square
    - Is that so unreasonable?

# Make every output a square

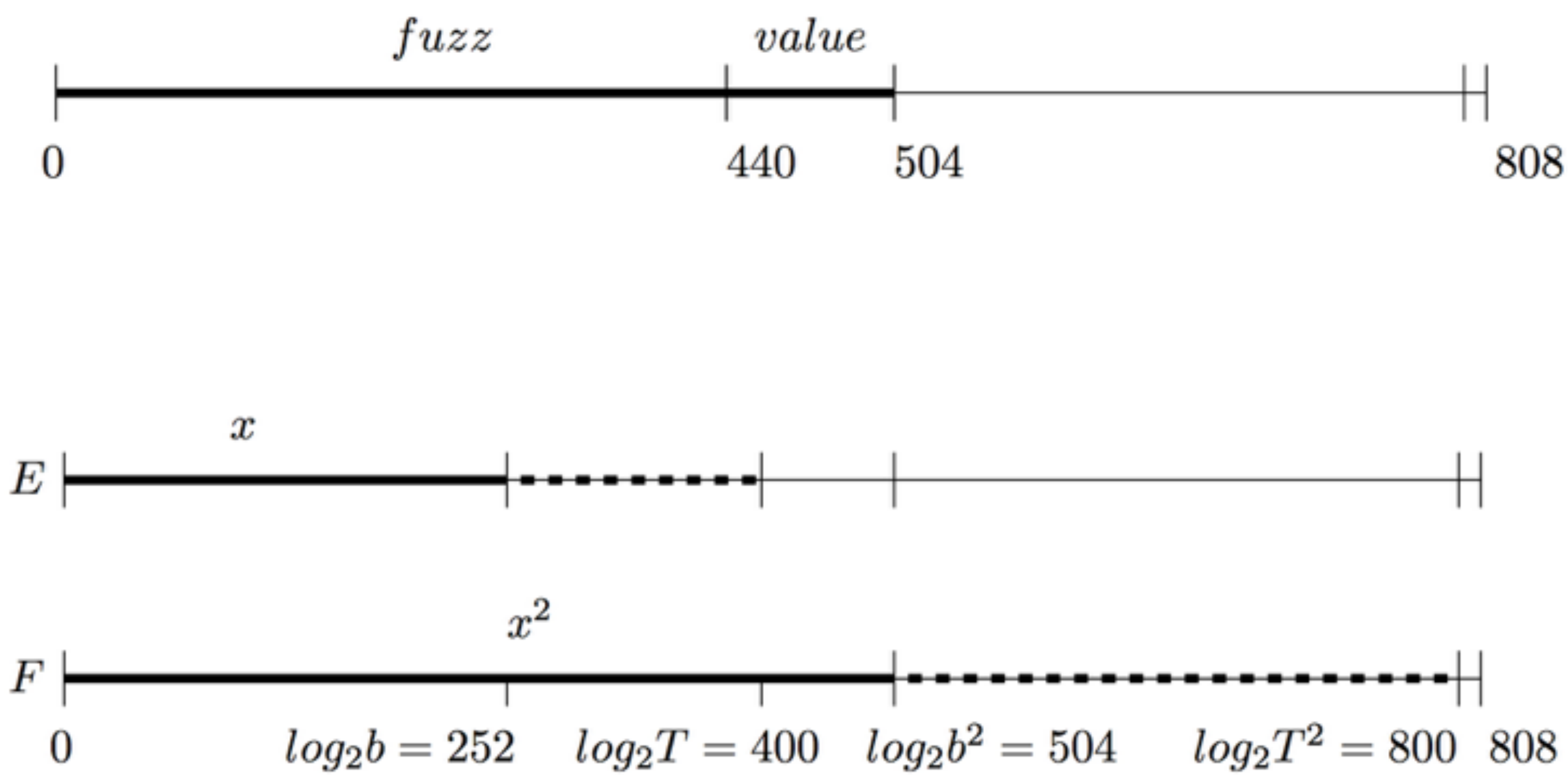


- $x' = value * 2^{fuzzbits} + U(0, 2^{fuzzbits})$
- $x = isqrt(x')$, $E = x*G$, $F = x*E$
- $delta = x' - x^2$
- How big is delta and what to do with it?
  - In a transaction, we can flush it into the fee
    - $Sum(output_j) + (fee)$
    - $Sum(F_j) + (Sum(delta_j) + random)$

# Maintaining secrecy

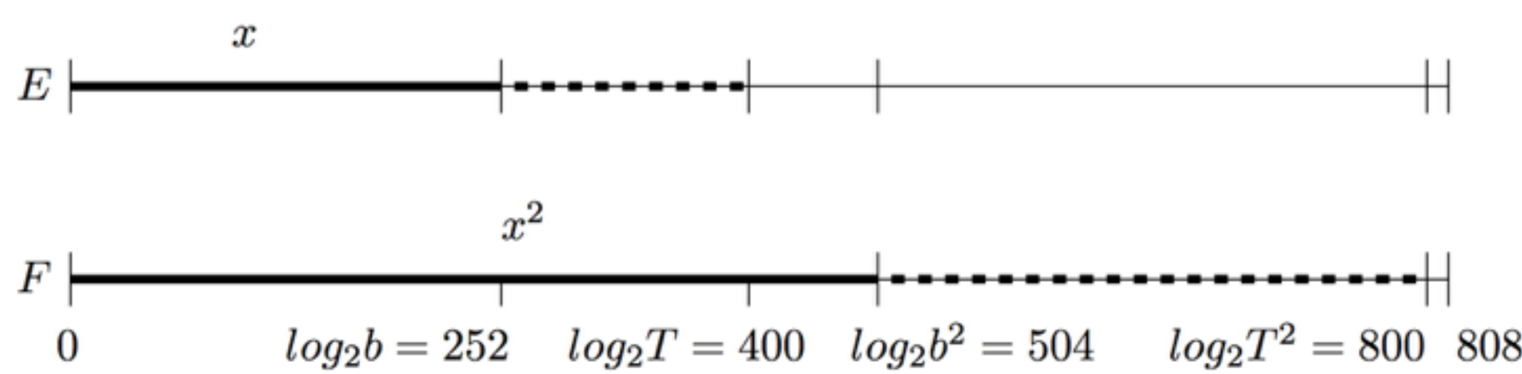
- If F is always the encryption of a square
  - E always contains half the 220 fuzz bits
  - That's only 55 bits of added security
  - We'll need 440 fuzz bits + 52 value bits for $x^2$
  - And CFT will need more

# A bigger curve




# Square and interval proof

- Knowing x in [0, b] and two large curves with base point G
- Proving x in some much wider [-T, T], $T = b*2^{t+l}$
- Commit to E=x*G, F = x*E
- Pick r, w in [0, T], commit to U = r*G, V = r*E, W = w*G
- c = HASH(E|F|U|V|W)
- m = r + c*x; q = w + c*(x+1)
- Verifier only needs (E, F, m, q, c)
- Check c*b < m < T, c*b < q < T
- Check c = HASH(E|F|m*G-c*E|m*E-c*F|q*G-c*(E+G))



# Compact Confidential Transactions

- Space efficient: Only 0.35kB per output
  - 102 bytes for each E, F; 50 for m, q; 16 for c; 32 for DH(x)
  - Compared to 2.5kB for CT, but CCT hides twice as many bits
  - Only store F in unspent outputs (UTXO)
- Semi computationally efficient: 60 output verifications/sec
  - 4 ECC 808-bit multiplications, faster because scalars are small
  - OpenSSL w/precalc on single core of a Q9550 “Core 2 Quad”
  - Good enough for real-time Bitcoin txns, but not for initial sync

# CT/CCT Comparison

| Metric | CT | CCT | Improvement |
|---|---|---|---|
| value bits hidden | 32 | 64+ | 100% |
| blockchain space, kB | 2.55 | 0.35 | 728% |
| verifications per second | 1300 libsecp256k1 | 600* OpenSSL | -53% |

(*normalised by 1.82x for published i7 CPU, can go a whole lot faster)

# CT/CCT Features

- Compatible with CoinJoin and variable denominations
- Compatible with spent transaction pruning
- Optional dual keys for an address
  - Spend keys unaffected, script language untouched
  - View private key provides visibility, but not spend power
  - View public key included in address
    - Optionally in scriptSig for link-ability
- Adjustable security parameters
- No way of identifying dust, no brain-wallets (which lack entropy anyway)

# Implementation

- PoW P2P blockchain and GUI
  - 6000 lines of Python
- Smallness prover/verifier only 60 lines
- CCT transaction handling only 400 lines
- Beware Python's "math.sqrt" and "**0.5"
  - They do not work for large numbers

# Work in progress

- Peer review
- Practical fee calculations for private and public chains
    - Sum-of-3 squares (3x more expensive) for zero leakage
- Faster multiplication
    - Reduce curve size requirement, scalars
    - Investigate curve extensions (GLV-GLS)
    - Implement faster algorithms (point halving, etc, hardware)
- Mitigate DoS attacks on slow computation

# Acknowledgements


- Thanks for significant input:
  - Andrew Poelstra
    - Broke an initial over-optimistic proof
    - Suggested statistical attack on m
  - Jochen Hoenicke
    - Found missing items in hash for combined proof
    - Suggested single-square is good enough
  - Jonathan Bootle
    - Suggested known group order attack on m
  - Gregory Maxwell
    - Review